

# Guides

## Getting Started

### Getting started extending Piwik

*Setup your development environment and learn the basics of Piwik plugin/theme development.* [Learn more »](#)

### Getting started using the Reporting API

*The Reporting API tutorial in less than one minute.* [Learn more »](#)

## In-depth guides

### All about Analytics Data

*Learn about how analytics reports are calculated and stored in Piwik (the Archiving Process) and how plugins can define their own reports.* [Learn more »](#)

### All about Tracking

*Learn in detail about how Piwik's tracking system works.* [Learn more »](#)

### MVC in Piwik

*Learn how Piwik handles HTTP requests and how Piwik generates the HTML that is displayed to the user.* [Learn more »](#)

## Piwik HTTP APIs

### Piwik's Reporting API

*Learn how Piwik exposes API methods in its Reporting API and how third party applications can use its Tracking API.* [Learn more »](#)

## Querying the Reporting API

*Learn how to query for report data via HTTP requests and from within Piwik's source code.* [Learn more »](#)

## Piwik Development

### Theming

*Learn how themes can change Piwik's look and feel and how you can create your own.* [Learn more »](#)

### Security in Piwik

*Learn how Piwik protects itself from potential attackers and how you can write secure code for your plugin.* [Learn more »](#)

### Internationalization

*Learn how Piwik makes its text available in many different languages and how your plugin can do the same.* [Learn more »](#)

### Automated Tests

*Learn how to setup unit, integration and UI tests for your new plugin and how to work with Piwik Core's tests.* [Learn more »](#)

### Manual UI Testing

*Learn how to manually test every part of the Piwik UI to ensure your code did not result in regressions.* [Learn more »](#)

## Piwik's UI

### Visualizing Report Data

*Learn about the different ways Piwik can display analytics data and how plugins can create new ways to display it.* [Learn more »](#)

### Working with Piwik's UI

*Learn about Piwik's JavaScript code and how to write JavaScript for your plugin.* [Learn more »](#)

## Piwik Configuration

### Piwik configuration

*Learn how Piwik is configured and how plugins can define their own configuration settings.* [Learn more »](#)

## Distributing your plugin

### Distributing your plugin

*Learn how to share your completed plugin with all other Piwik users. Learn about the Piwik marketplace and how you can use it to distribute your plugin.* [Learn more »](#)

## Piwik Internals

### Piwik's extensibility points

*Learn about everything that allows Piwik to be extended including, the eventing system, report metadata, scheduled tasks and, of course, plugins themselves.* [Learn more »](#)

## Persistence & the MySQL Backend

*Learn about what log data and analytics data consists of and about how it is stored in Piwik's MySQL database.* [Learn more »](#)

## Piwik on the command line

*Learn about Piwik's command line tool and how your plugin can extend it.* [Learn more »](#)

## About the Piwik Project

### Contributing to Piwik Core

*Learn how to contribute changes to Piwik Core. Learn about Piwik's coding standards and the contribution process.* [Learn more »](#)

## Core Team Workflow

*Learn how the core development team manages Piwik development and makes changes to the source code. Learn how you can participate in this process. [Learn more »](#)*

# Getting started extending Piwik

---

So you're a Piwik user and you need Piwik to do something it doesn't do. Or maybe you're a user of another web app and you want to integrate it with this amazing analytics software you've heard so much about (Piwik, naturally). Well, you've come to the right place!

This guide will show you:

- **how to get your development environment setup,**
- **how to create a new Piwik Plugin,**
- **and where to go next to continue building your extension.**

### Guide Assumptions

This guide assumes that you, the reader, can code in PHP and can setup your own local webserver. If you do not have these skills, you will not be able to successfully understand this guide.

There are many resources on the internet you can use to learn PHP. We recommend reading [these tutorials](#).

## Piwik Plugins

Pretty much all the functionality you see when you use Piwik is provided through **Piwik Plugins**. The core of Piwik (unsurprisingly termed **Piwik Core**) contains the tools that the plugins use to provide this functionality. If you wanted to add more functionality to Piwik you'd create your own plugin and distribute it on the [Piwik Marketplace](#) so other users could use it.

*Note: If you want to integrate another piece of software with Piwik and you only need to know how to track visits or how to retrieve reports, read more about the [Tracking API](#) and the [Reporting API](#).*

## What's possible with Piwik plugins?

Here are some of the things you could accomplish through your own plugin:

- track new visitor data and create new reports that aggregate the new data
- track third party data and display it in new reports
- show existing reports in novel new ways
- send scheduled reports through new mediums or in new formats

These are only a few of the possibilities. Many of the existing plugins do things that cannot be categorized in such a way. For example, the [Annotations](#) plugin lets users create save notes for dates without needing any modifications to **Piwik Core**. The [DBStats](#) plugin will show users statistics about their MySQL database. The [Dashboard](#) plugin provides a configurable way to view multiple reports at once.

**Whatever ideas your imagination cooks up, we think it's highly probable you can implement them with Piwik.**

## What's not possible with Piwik plugins?

Well, we're not really sure. It might be hard to get your idea to scale, or maybe your idea involves creating some hardware that you have to distribute, but these problems can all be overcome in some way.

**Right now, we think *anything* is possible with Piwik, and we want YOU to prove us right!**

## Getting setup to extend Piwik

### Get the appropriate tools

Before we start extending Piwik, let's make sure you have the tools you'll need to do so. Make sure you have the following installed:

- **A PHP IDE or a text editor.** We who work on Piwik recommend you use [PHPStorm](#), a very powerful IDE built specifically for developing in PHP.
- **A webserver**, such as [Apache](#) or [Nginx](#).
- [git](#) so you can work with the latest Piwik source code.
- [composer](#) so you can install the PHP libraries Piwik uses.
- **A browser**, such as [Firefox](#) or [Chrome](#). Ok, so this you've probably got.

The following tools aren't required for this guide, but you may find them useful when you continue writing your plugin:

- [PHPUnit](#) Necessary if you want to write or run automated tests.
- [xhprof](#) If you'd like to profile your code and debug any inefficiencies. See also [our guide for installing xhprof](#).

- [python](#) If you're going to be doing something with the log importer.

## Get & Install Piwik

Now that you've got the necessary tools, you can install Piwik. First, we'll get the very latest version of Piwik's source code using git.

Open a terminal and

```
cd
```

into the directory where you want to install Piwik. Then run the following command:

```
$ git clone https://github.com/piwik/piwik piwik
```

Then run the following command to install some third party libraries:

```
$ composer.phar install
```

Now that you've got a copy of Piwik, you'll need to point your webserver to it. The specific instructions for configuring your webserver depends on the webserver itself. You can see instructions for Apache[here](#) and instructions for Nginx [here](#).

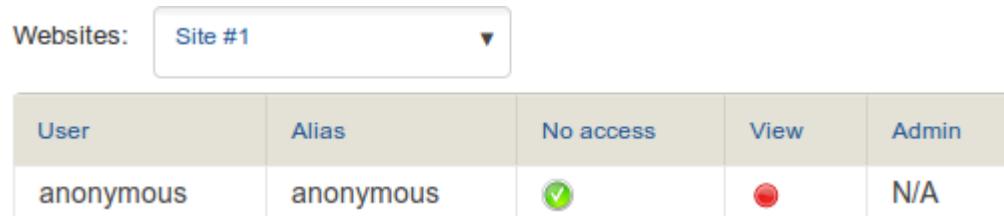
Once your webserver is configured, load Piwik in your browser by going to the URL

```
http://localhost/
```

. Complete the installation process by following the instructions presented to you.

### Adding anonymous access to your reports

Before we finish, we're going to allow anyone to view reports on your new Piwik environment. Open the *Manage > Users* admin page and click the red icon in the **View** column for the **anonymous** user:



The screenshot shows a table with columns: User, Alias, No access, View, and Admin. The 'User' row for 'anonymous' has a green checkmark in the 'View' column and a red circle with a white dot in the 'Admin' column. The 'Alias' column also contains 'anonymous'. The 'No access' column has a grey circle with a checkmark.

User	Alias	No access	View	Admin
anonymous	anonymous	✓	✗	N/A

This will make it possible to view raw report data without having to supply a **token\_auth**.

### Installing the VisitorGenerator plugin

Finally, you'll want to install a plugin that will help with development. [Download the VisitorGenerator plugin here](#) and install it by extracting the archive's contents to your Piwik's **plugins** subdirectory.

## Add some test data

You've got the necessary tools and Piwik itself. You're ready to create your first plugin now, but before we do that, let's add some test data for you to play with.

In your browser load Piwik and navigate to *Settings > Plugins > Manage*. Look for the *Visitor Generator* plugin and enable it. Then on the admin menu to the left, click on *Visitor Generator* (under *Diagnostic*).

On this page you'll see a form where you can select a site and enter a number of days to generate data for:

Choose website	<span style="margin-right: 10px;">Site #1</span> ▾
Days to compute	3

You are about to generate fake visits which will be recorded in your Piwik database. It will **not** be possible to easily delete these visits from the piwik logs.

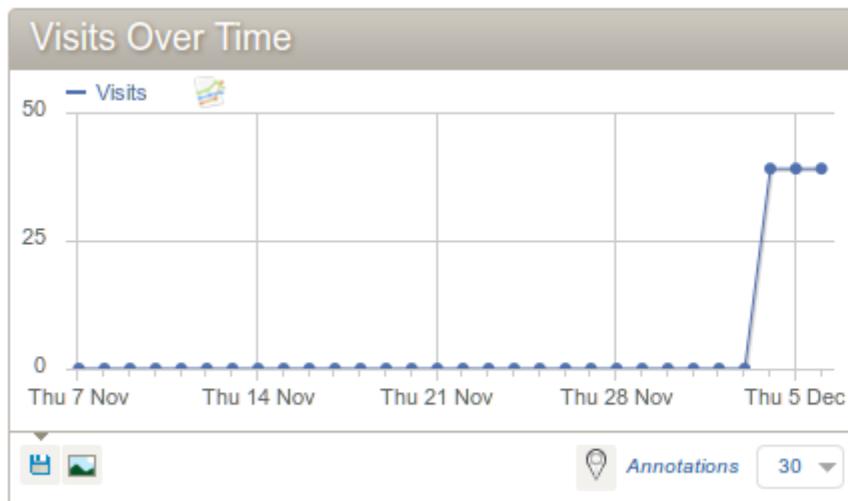
**This will generate approximately 181 fake actions on this site for each day.**

Are you sure you want to generate fake visits?

Yes, I am sure!

Let's generate data for three days. Enter **3** in the **Days to compute** field, check the **Yes, I am sure!** checkbox and click **Submit**.

Now click on the **Dashboard** link at the top of the screen. You should see that reports that were previously empty now display some statistics:



### Visitors in Real-time

Date	Visits	Pageviews
Last 24 hours	39	121
Last 30 minutes	0	0
Thu 5 Dec - 23:20:28 (3 min 28s)    -   - IP: 13.5.111.3		
from <a href="#">Google</a> - "piwik bingbot" # 4		
Pages:		
Thu 5 Dec - 23:19:45 (17 min 40s)    -   - IP: 13.5.111.3		
from <a href="#">demo.piwik.org</a>		

### Support Piwik!

Piwik will always cost you nothing, but it means it costs us nothing to make it free.

**Piwik needs your continued support to thrive.**

If you feel that Piwik has added value to your business or endeavour, **please consider supporting us**.

**How much is Piwik worth to you?**

(Click on the slider to select an amount, then click the "Subscribe" button)



**Subscribe**



[Make a one-time donation](#)

This widget is only displayed to you because you are a Piwik user.

### Keywords

[Keyword](#)

can direct download be tracked

## Create a plugin

Your development environment is setup, and you are now ready to create a plugin! Creating a plugin consists primarily of creating a couple files. Piwik comes with a handy command-line tool that will do this legwork for you. In the root directory of your Piwik install, run the following command:

```
./console generate:plugin --name="MyPlugin"
```

Replace **MyPlugin** with the name of your plugin (for example, **UserSettings** or **DevicesDetection**). When the tool asks if it should create an API & Controller, enter

```
'y'
```

In your browser load Piwik and navigate to *Settings > Plugins > Manage*. Look for your plugin in the list of plugins, you should see it disabled:

MyPlugin	0.1.0	My new plugin. ( <a href="#">GPL v3 or later</a> ) By <a href="#">Piwik</a> .
-	-	Un

## Plugin directory structure

The command-line tool will create a new directory for your plugin (in the **plugins** sub-directory) and fill it with some files and folders. Here's what these files and folders are for:

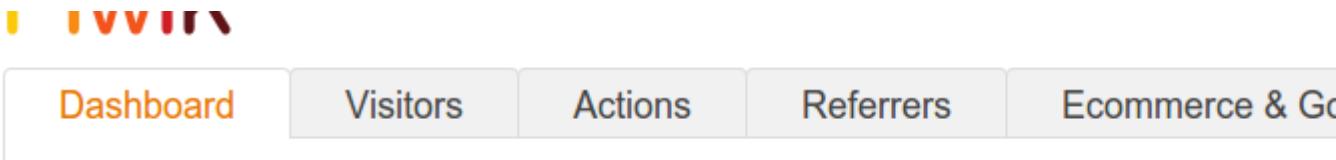
- **API.php**: Contains your plugin's API class. This class defines methods that serve data and will be accessible through Piwik's [Reporting API](#).
- **Controller.php**: Contains your plugin's Controller class. This class defines methods that generate HTML output.
- **MyPlugin.php**: Contains your plugin's Plugin Descriptor class. This class contains metadata about your plugin and a list of event handlers for Piwik events.
- **plugin.json**: Contains plugin metadata such as the name, description, version, etc.
- **README.md**: A dummy README file for your plugin.
- **javascripts**: This folder is where you'll put all your new JavaScript files.
- **screenshots**: TODO: ???
- **templates**: This folder is where you'll put all your [Twig](#) templates.

## Make your plugin do something

For this guide we don't want to do anything really complicated, so we'll just define a new analytics report that uses realtime data and add a new page to Piwik that displays the report. We'll also add some settings so the data the report displays will change.

## Adding a new reporting page

First, let's add a new reporting page to Piwik. Links to Piwik's reporting pages are displayed on the main page under the logo:



What gets put there is determined by plugins that add menu items through the [MenuMain](#) class.

## Adding a controller method

To add a new page, we'll start by creating the page itself. In Piwik, HTML is generated and served by [Controller](#) methods, so we'll add a new method to your plugin's controller. Since the HTML generated by this method will be the main page for this plugin, we'll call the method **index**.

```
class Controller extends \Piwik\Plugin\Controller

{
    // the output of this method will be accessible via a URL like
    // index.php?module=MyPlugin&action=index

    public function index()
    {
        //
    }
}
```

## Using a View

Inside the Controller method we'll generate HTML by using a [View](#). View objects manage Twig templates; they define some basic properties, [filters](#) and [Twig functions](#) for templates to use, and allow you to set other template properties.

Right now, we don't have much to display, so we won't be setting any properties. In your controller method, add the following code (replacing

```
'MyPlugin'
with the name of your plugin):
$view = new View("@MyPlugin/index.twig");
return $view->render();
```

### Template Naming Convention

Our template will be named **index.twig** since the method it's in is named **index**. This is the naming convention used in Piwik. If a template doesn't correspond to a controller method, its name should describe what it outputs and be prefixed with an underscore (for example, **\_dataTable.twig**).

## Adding a Twig template

The **console** tool will automatically create a Now we'll create our Twig template. In the **templates** subdirectory of your plugin's root directory, add a file named **index.twig**. In the file, add the following code:

```
<h1>Realtime Analytics</h1>
```

```
<strong>Hello world!</strong>
```

If you open a browser and open the URL `http://localhost/index.php?module=MyPlugin&action=index&idSite=1&date=to day&period=day` after replacing `MyPlugin` with the name of your plugin, you should see the page you just created!

# Realtime Analytics

## Hello world!

### Adding a menu item

Now that there's a page, we need to add it to the reporting menu so users can get to it. The [MenuMain](#) class (and other menu managing classes) allows plugins to add menu items through an event named [Menu.Reporting.addItem](#).

#### About Events

Events are one of the main ways Piwik allows plugins to add new functionality. At certain points during execution, Piwik will post an event to the `EventDispatcher`. Plugins can register callbacks with events so those callbacks will be called when events are posted. Our plugin has to handle this event, so we'll associate a method with the event. In the `getListHooksRegistered` method of your plugin descriptor class (the class that extends [Piwik\Plugin](#) and has the same name as your plugin), add the following code:

```
return array(
    'AssetManager.getJavaScriptFiles' => 'getJsFiles',
    'Menu.Reporting.addItem' => 'getReportingMenuItems'
);
```

Then add this method to the class:

```
public function getReportingMenuItems()
{
    \Piwik\Menu\MenuMain::getInstance()->add(
        $category = 'General_Visitors', // this is a 'translation token'. it will be replaced by
```

```

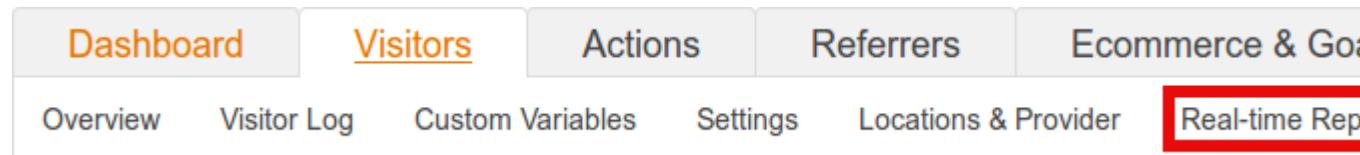
        // a translated string based
        // on the user's language preference.

        // read about internationalization below to learn more.

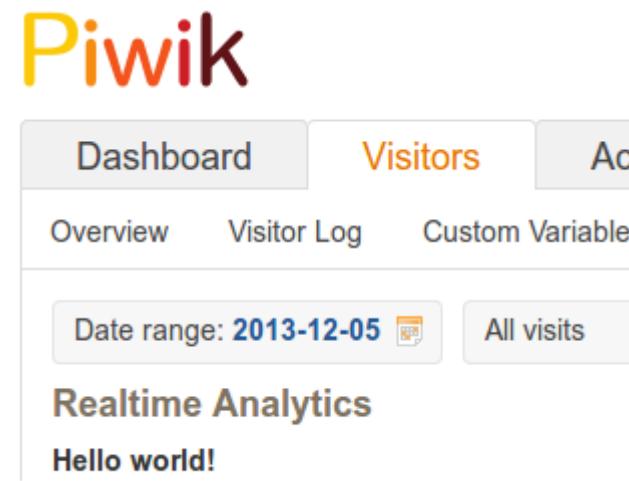
        $title = 'Real-time Reports',
        $urlParams = array('module' => $this->getPluginName(), 'action' => 'index')
    );
}

```

Now, if you load Piwik in a browser, you'll see the menu item:



If you click on it, the page will be loaded below the period selector:



## Adding a new report

We've created a plugin and got it to display something in Piwik's UI. Now let's make it show something useful. We're going to create a new report that uses the realtime visit data returned by the [Live!plugin](#) and reports on the browsers used.

### On reports and metrics

Reports and metrics are the two types of analytics data Piwik calculates and stores. Metrics are just single values, like **visits**. Reports are two dimensional arrays of values, usually metric values and are stored using the [DataTable](#) class.

## Adding an API method

Reports and metrics are all served through API class methods, so we'll add a new one for our report. In your plugin's API class (stored in **API.php**), add the following method:

```
public function getLastVisitsByBrowser($idSite, $period, $date,  
    $segment = false)  
{  
    return array();  
}
```

### Analytics Parameters

Every API method that serves a report or metric will have the parameters listed above. This is because all analytics data describes data that is tracked for a certain website and during a certain period. A [segment](#) can be supplied to further reduce the data that is analyzed, but it is optional (which is why the parameter defaults to **false**).

).

The website is determined by the

**\$idSite**

parameter and the period by both the

**\$period**

and

**\$date**

parameters. The segment is determined by the value in the

**\$segment**

parameter.

You can see the output of this method if you visit this

URL: <http://localhost/index.php?module=API&method=MyPlugin.getLastVisitsByBrowser&idSite=1&date=today&period=week>(remember to replace **MyPlugin** with the name of your plugin).

## Implementing the API method

Our new report will use realtime visit data, so the first thing our API method must do is get it. We'll use the **Live.getLastVisitsDetails** method:

```
public function getLastVisitsByBrowser($idSite, $period, $date,  
    $segment = false)  
{  
    $data = \Piwik\Plugins\Live\API::getInstance()->getLastVisitsDetails(
```

```

        $idSite,
        $period,
        $date,
        $segment,
        $numLastVisitorsToFetch = 100,
        $minTimestamp = false,
        $flat = false,
        $doNotFetchActions = true
    );
    $data->applyQueuedFilters();

    return array();
}

```

This will return a [DataTable](#) instance that holds information for each visit in its rows.

#### About [DataTable](#)

As stated above [DataTable](#)s store report data. They are essentially just an array of rows, where each row is an array of columns.

Now that we've got a list of visits, we need to count the number of visits for each browser used. We'll do this by manually iterating through each row to create a new [DataTable](#) instance:

```

public function getLastVisitsByBrowser($idSite, $period, $date,
    $segment = false)

{
    $data = \Piwik\Plugins\Live\API::getInstance()->getLastVisitsDetails(
        $idSite,
        $period,
        $date,
        $segment,
        $numLastVisitorsToFetch = 100,
        $minTimestamp = false,
        $flat = false,

```

```

$doNotFetchActions = true

);

$result = $data->getEmptyClone($keepFilters = false); // we
could create a new instance by using new DataTable(),
// but tha
t wouldn't copy DataTable metadata, which can be
// useful.

foreach ($data->getRows() as $visitRow) {
    $browserName = $visitRow->getColumn('browserName');

    $resultRowForBrowser = $result->getRowFromLabel($browse
rName);

    // if there is no row for this browser, create it
    if ($resultRowForBrowser === false) {
        $result->addRowFromSimpleArray(array(
            'label' => $browserName,
            'nb_visits' => 1
        ));
    } else { // if there is a row, increment the visit count
        $resultRowForBrowser->setColumn('nb_visits', $result
RowForBrowser->getColumn('nb_visits') + 1);
    }
}

return $result;
}

```

If you  
visit <http://localhost/index.php?module=API&method=MyPlugin.getLastVisitsByBro
wser&idSite=1&date=today&period=weekly> you should see the new report!

## Realtime Reports vs Archived Reports

This new API method directly accesses visit data. That is because the report is a real-time report. Most reports aren't in real-time because the amount of time it would take to process the visits they report on would make the UI unusable. These reports are calculated and **cached** during the [Archiving Process](#). To learn more, read our [All About Analytics Data](#) guide.

## Displaying the report

Now that we've defined a new report, we need to display it. We'll do this by adding a new method to our controller:

```
public function getLastVisitsByBrowser()  
{  
    // ...  
}
```

We add a new method because we'll be using a special view class that will sometimes use AJAX to reload the report, so there has to be a way to get **just** the HTML for the display.

The special view class we'll use is called [ViewDataTable](#), and here's how we'll use it:

```
public function getLastVisitsByBrowser()  
{  
    // ViewDataTable instances are created by the Factory, not t  
    // hrough the new operator  
  
    $view = \Piwik\ViewDataTable\Factory::build(  
        $defaultVisualization = 'table',  
        $apiAction = 'MyPlugin.getLastVisitsByBrowser' // rememb  
        // er to replace MyPlugin with the name of your plugin  
    );  
  
    // after a ViewDataTable instance is created, it must be con  
    // figured so the display is perfect  
    // for the report. this is done by setting properties of the  
    // ViewDataTable::$config object.  
    $view->config->show_table_all_columns = false;  
    $view->config->show_goals = false;
```

```
$view->config->translations['label'] = 'Browser';

return $view->render();
}
```

## Report Visualizations

The [ViewDataTable](#) class outputs what Piwik calls a **report visualization**. Report visualizations display an analytics report in some way. They can be in any format, including HTML or JavaScript (like the default **table** visualization or one of the graphs) or an image (like the **sparkline** visualization).

Plugins can create their own visualizations. To find out how, read our [Visualizing Report Data](#) guide (after you're done with this guide, of course).

Now that we have a method that outputs a display for the report, we need to embed it in the plugin's main page. Change the

```
index()  
controller method to look like this:  
public function index()  
{  
    $view = new View("@MyPlugin/index.twig");  
    $view->getLastVisitsByBrowserReport = $this->getLastVisitsByBrowser();  
    echo $view->render();  
}
```

And change the **index.twig** template to look like this:

```
<h1>Realtime Analytics</h1>  
  
{ { getLastVisitsByBrowserReport|raw } }
```

Now if you view the page in Piwik, you'll see something like this:

## Updating the report in realtime

So now there's a page with a report that displays the browsers of the latest visitors. It uses realtime data, but it's not truly realtime since after a couple minutes, the report will be out

of date. To make the report more realtime we'll add an option for the user to update the report.

We'll allow the user to update the report by clicking a link. Add the following to the bottom of your `index.twig` file:

```
<a id="realtime-reports-reload" href="#">Reload</a>
```

## Adding JavaScript files to Piwik

To make the report reload itself, we'll have to write some JavaScript code. This means we'll need a JavaScript file.

The

```
console
```

command line tool will automatically generate a JavaScript file called `plugin.js` which we'll use. If it didn't exist, though, we'd have to create a new file in the `javascripts` subdirectory and let Piwik know about through the [AssetManager.getJavaScriptFiles](#) event.

## Reloading the report

In your `plugin.js` file, add the following code in the

```
$(document).ready
  callback:
    // Piwik Loads most content via AJAX, so we use $.on instead of
    // $.click directly
    $('body').on('click', '#realtime-reports-reload', function (e)
    {
      e.preventDefault();

      var $dataTableRoot = $('div.dataTables[data-report="MyPlugin.getLastVisitsByBrowser"]');
      // in the UI, the root element of a displayed report has a JavaScript object associated with it.
      // we can use this object to reload the report.
      var dataTableInstance = $dataTableRoot.data('uiControlObject');
```

```

    // we want the table to be completely reset, so we'll reset
    // some query parameters then reload

    // the report
    dataTableInstance.resetAllFilters();
    dataTableInstance.reloadAjaxDataTable();

    return false;
});

```

If you click on the new **Reload** link, you'll see the report being reloaded.

Well, our simple plugin is done! It defines a new report, displays it and makes sure the data it displays is fresh. But we can do better! The next two sections will show you how.

## Making the report configurable

The report we've defined is interesting, but we could easily aggregate on another visit property. For example, the report could be **getLastVisitsByScreenType** or **getLastVisitsByCity**. In this section, we're going to make it possible for users to change what the report displays.

### Creating a plugin setting

We'll create a **plugin setting** which will control which visit property the plugin uses to generate our report. The first step is to create a **Settings** class:

```

<?php

namespace Piwik\Plugins\MyPlugin; // remember to rename MyPlugin
// with the name of your plugin

class Settings extends \Piwik\Plugin\Settings
{
    protected function init()
    {
        // ...
    }
}

```

```
}
```

Put this class in a file called **Settings.php** in your plugin's root directory.

The **Settings** class is a special class that is automatically detected by Piwik. Piwik uses the information it sets to add a new section for your plugin in the *Plugins > Settings* admin page.

We're going to create one setting that can be set differently by each user. First, let's think about our new setting. It's going to determine the column of the **Live.getLastVisitsDetails** that we'll aggregate by. So it's a string and has a limited number of valid values. We'll use a single select dropdown (just a normal

```
<select>
```

) for it.

Now, let's add an attribute and new method for this setting:

```
class Settings extends \Piwik\Plugin\Settings
{
    public $realtimeReportDimension;

    protected function init()
    {
        $this->realtimeReportDimension = $this->createRealtimeReportDimensionSetting();
        $this->addSetting($this->realtimeReportDimension);
    }

    private function createRealtimeReportDimensionSetting()
    {
        // ...
    }
}
```

Then we'll implement the

```
createRealtimeReportDimensionSetting
```

method:

```
private function createRealtimeReportDimensionSetting()
```

```

{
    $setting = new \Piwik\Settings\UserSetting('reportDimension',
        'Report Dimension');

    $setting->type = self::TYPE_STRING;

    $setting->uiControlType = self::CONTROL_SINGLE_SELECT;

    $setting->description = 'Choose the dimension to aggregate by';

    $setting->availableValues = MyPlugin::$availableDimensionsForAggregation; // replace 'MyPlugin'!

    $setting->defaultValue = 'browserName';

    return $setting;
}

```

Notice how

`$settings->availableValues`

is set to

`MyPlugin::$availableDimensionsForAggregation`

. The `availableValues` property should be set to an array mapping column values with their appropriate display text. This array will probably come in handy later, so we'll stash it in our plugin descriptor class.

In your plugin descriptor class add the following code:

```

public static $availableDimensionsForAggregation = array(
    'browserName' => 'Browser',
    'visitIp' => 'IP',
    'visitorId' => 'Visitor ID',
    'searches' => 'Number of Site Searches',
    'events' => 'Number of Events',
    'actions' => 'Number of Actions',
    'visitDurationPretty' => 'Visit Duration',
    'country' => 'Country',
    'region' => 'Region',
    'city' => 'City',
    'operatingSystem' => 'Operating System',
    'screenType' => 'Screen Type',
)

```

```

'resolution' => 'Resolution'

// we could add more, but let's not waste time.

);

```

If you go to the *Plugins > Settings* admin page you should see this new setting:

## MyPlugin

---

The screenshot shows a settings page for 'MyPlugin'. At the top, there is a section labeled 'Report Dimension' with the sub-instruction 'Choose the dimension to aggregate by'. A dropdown menu is open, showing 'Browser' as the selected option. Below the dropdown, the text 'Default browser' is visible. At the bottom left of the page is a large, rounded rectangular button with the word 'Save' in white text.

## Using the new setting

To use the setting, first we need to get the setting value in our API method and then aggregate using it. Change your API method to look like this:

```

public function getLastVisitsByBrowser($idSite, $period, $date,
    $segment = false)

{
    // get realtime visit data

    $data = \Piwik\Plugins\Live\API::getInstance()->getLastVisitsDetails(
        $idSite,
        $period,
        $date,
        $segment,
        $numLastVisitorsToFetch = 100,
        $minTimestamp = false,
        $flat = false,
        $doNotFetchActions = true
    );
}

```

```
);

$data->applyQueuedFilters();

// read the setting value that contains the column value to aggregate on

$settings = new Settings('MyPlugin');

$columnName = $settings->realtimeReportDimension->getValue();

// count visits to create our result

$result = $data->getEmptyClone($keepFilters = false); // we could create a new instance by using new DataTable(),
// but that wouldn't copy DataTable metadata, which can be
// useful.

foreach ($data->getRows() as $visitRow) {

    $columnValue = $visitRow->getColumn($columnName);

    $resultRowForBrowser = $result->getRowFromLabel($columnValue);

    i// if there is no row for this browser, create it

    if ($resultRowForBrowser === false) {

        $result->addRowFromSimpleArray(array(
            'label' => $columnValue,
            'nb_visits' => 1
        ));

        } else { i// if there is a row, increment the visit count

            $resultRowForBrowser->setColumn('nb_visits', $resultRowForBrowser->getColumn('nb_visits') + 1);
        }
    }
}
```

```

    }

    return $result;
}

```

Now we'll want to make sure the column heading in the report display has the correct text. Right now, it will display **Browser** no matter what the setting value is:

The screenshot shows the Realtime Analytics interface. At the top, there are two input fields: 'Date range: 2013-12-05' with a calendar icon and 'All visits' with a dropdown arrow. Below this, the title 'Realtime Analytics' is displayed. A table follows, with columns labeled 'Browser' and 'Visits'. The data shows two entries: 'France' with 9 visits and 'China' with 5 visits.

Browser	Visits
France	9
China	5

Change the **getLastVisitsByBrowser** controller method to the following:

```

public function getLastVisitsByBrowser()
{
    // ViewDataTable instances are created by the Factory, not through the new operator
    $view = \Piwik\ViewDataTable\Factory::build(
        $defaultVisualization = 'table',
        $apiAction = 'MyPlugin.getLastVisitsByBrowser' // remember to replace MyPlugin with the name of your plugin
    );

    // after a ViewDataTable instance is created, it must be configured so the display is perfect
    // for the report. this is done by setting properties of the ViewDataTable::$config object.
    $view->config->show_table_all_columns = false;
    $view->config->show_goals = false;

    $settings = new Settings('MyPlugin');
}

```

```

$columnToAggregate = $settings->realtimeReportDimension->get
tValue();

$columnNameLabel = MyPlugin::$availableDimensionsForAggregation
[$columnToAggregate]; // remember to replace MyPlugin with the
name of your plugin

$view->config->translations['label'] = $columnNameLabel;

return $view->render();
}

```

View the report now and you'll see:

The screenshot shows the Piwik Realtime Analytics interface. At the top, there is a date range selector set to '2013-12-05' and a dropdown menu set to 'All visits'. Below this, the title 'Realtime Analytics' is displayed. A table follows, with columns labeled 'Country' and 'Visits'. The data shows two entries: 'France' with 9 visits and 'China' with 5 visits.

Country	Visits
France	9
China	5

## Rename the report

Finally, we'll rename the report. After all, it can do more than just aggregate the last 100 visits by browser now. Rename all occurrences of `getLastVisitsByBrowser` with `getLastVisitsByDimension`. Make sure you replace it in the following files:

- API.php
- Controller.php
- plugin.js
- index.twig

## Internationalizing your plugin

The other improvement we'll make to our plugin is to use Piwik's [internationalization](#) system so our plugin can be made available in multiple languages.

Internationalization is achieved in Piwik by replacing translated text, like

```
"Realtime Analytics"  
, with unique identifiers, like  
"MyPlugin_RealtimeAnalytics"  
called translation tokens.
```

Translation tokens are associated with translated text in multiple JSON files, one for each supported language. In code, the translation tokens are converted into translated text based on the user's selected language.

## Creating a language file

To internationalize our plugin, first, we'll create an english language file to hold our translated text. In your plugin's root directory, create a subdirectory named **lang**. In that folder, create a file named **en.json** and put the following in it (replace **MyPlugin** with the name of your plugin):

```
{  
  "MyPlugin": {  
  
  }  
}
```

We're going to move all of the translated text in our plugin to this file.

## Internationalizing our Twig Template

Translation tokens are translated in templates via the [translate](#) filter. We only use one piece of translated text in our template:

```
"Realtime Analytics"  
. First, we'll add an entry for this in the en.json file we just created. We'll use  
the RealtimeAnalytics translation token:
```

```
{  
  "MyPlugin": {  
    "RealtimeAnalytics": "Realtime Analytics"  
  }  
}
```

Then replace the text in your template with the following:

```
<h1>{{ MyPlugin_RealtimeAnalytics|translate }}</h1>  
  
{{ getLastVisitsByDimensionReport|raw }}
```

## Internationalizing our setting

Now, let's internationalize the text we use in our **Settings** class. First, we'll add entries for the text we use:

```
{  
    "MyPlugin": {  
        "RealtimeAnalytics": "Realtime Analytics",  
        "ReportDimensionSettingDescription" : "Choose the dimension to aggregate by"  
    }  
}
```

Then we'll use the new translation token in the **createRealtimeReportDimensionSetting** method:

```
$setting->description = \Piwik::translate('MyPlugin_ReportDimensionSettingDescription'); // replace 'MyPlugin'!
```

We also need to internationalize the names of each possible setting value. We'll do this by using translated text in the

**MyPlugin::\$availableDimensionsForAggregation**

static variable. Of course, we can't call [Piwik::translate](#) when setting a static field, so we'll have to add a new method.

We're not going to add any translation tokens to our **en.json** file this time. This is because the translations already exist for core plugins. Replace the

**MyPlugin::\$availableDimensionsForAggregation**

field with this:

```
public static $availableDimensionsForAggregation = array(  
    'browser' => 'UserSettings_ColumnBrowser',  
    'visitIp' => 'General_IP',  
    'visitorId' => 'General_VisitorID',  
    'searches' => 'General_NbSearches',  
    'events' => 'Events_NbEvents',  
    'actions' => 'General_NbActions',  
    'visitDurationPretty' => 'VisitorInterest_ColumnVisitDuration',  
    'country' => 'UserCountry_Country',
```

```

    'region' => 'UserCountry_Region',
    'city' => 'UserCountry_City',
    'operatingSystem' => 'UserSettings_ColumnOperatingSystem',
    'screenType' => 'UserSettings_ColumnTypeOfScreen',
    'resolution' => 'UserSettings_ColumnResolution'

    // we could add more, but let's not waste time.

);

```

Then, add this method to your plugin's plugin descriptor class:

```

public static function getAvailableDimensionsForAggregation()
{
    return array_map(array('Piwik', 'translate'), self::$availableDimensionsForAggregation);
}

```

Finally in the `createRealtimeReportDimensionSetting` method, replace

```

MyPlugin::$availableDimensionsForAggregation
with
MyPlugin::getAvailableDimensionsForAggregation()
.
```

## Internationalizing report column headers

Since we already use translation tokens in the

```

MyPlugin::$availableDimensionsForAggregation
field, and the column headers are set using the same data, all we have to do is use
MyPlugin::getAvailableDimensionsForAggregation
in the getLastVisitsByDimension controller method:
$columTranslations = MyPlugin::getAvailableDimensionsForAggregation(); // remember to replace MyPlugin with the name of your
plugin
$columLabel = $columTranslations[$columToAggregate];

```

## What to read next

Ok! You've set up your development environment and created your plugin! Now all you have to do is make it do what you want. The bad news is that this is the hard part. The

good news is that we've written a bunch of other guides to help you shorten the learning curve.

**Note: Our guides are great, but if you learn better through examples or just don't want to do a lot of reading, why not check out our [tutorials](#)? We've written one for everything we think you might want to do.**

Based on what you want your plugin to accomplish, here's what you might want to read next:

- If you're interested in **creating new analytics reports**, you may want to read our [All About Analytics Data](#) and [Visualizing Report Data](#) guides.
- If you're interested in **changing the look and feel of Piwik**, read our [Theming](#) guide.
- If you're interested in **taking part in core development**, read our [Contributing to Piwik Core](#) guide.
- If you're interested in **integrating Piwik with another technology**, you might want to read our [All About Tracking](#) guide to learn how to use our Tracking API. You might also want to read our [Piwik's HTTP API](#) guide to learn about Piwik's Reporting API.
- If you'd like to **add new console commands**, read our [Piwik on the command line](#) guide.
- If you want to **use automated testing to ensure your plugin works**, read your [Automated Tests](#) guide.

And **make sure to read our security guide, [Security in Piwik](#)!** We have very high security standards that your plugin or contribution **must** respect.

When you've completed your plugin, you can read our [Distributing your plugin](#) guide to learn how to **share your plugin with other Piwik users**.

# Piwik API Tutorial: Get Your Top 10 Keywords

---

This tutorial will show you how easy it is to request the **yesterday's top 10 keywords in XML format**.

## Build the URL

To build the URL of the API call, you need:

- your Piwik base URL (replace `demo.piwik.org` with the URL and path of your Piwik server)

**`http://demo.piwik.org/?module=API`**

- the name of the method you want to call. It has the format `moduleName.methodToCall` (see the list on [API Methods](#)). You need to request the last keywords from the plugin Referers: the method parameter is:

**`method=Referers.getKeywords`**

- the website id.

**`idSite=1`**

- the date parameter. This can be *today*, *yesterday*, or any date with the format `YYYY-MM-DD`

**`date=yesterday`**

- the period parameter. This can be *day*, *week*, *month* or *year*

**`period=day`**

Alternatively, if you wanted to request all of the keywords from a given date, you could use a date range parameter. For example, to request all of the keywords since January 1st 2011:

**`**period=range&date=2011-01-01,yesterday**`**

- the format parameter. Defines the output format of the data: XML, JSON, CSV, PHP (serialized PHP), HTML (simple html)

**`format=xml`**

- (optional) the filter\_limit parameter that defines the number of rows returned

**`filter_limit=10`**

The final url

is [`http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&date=yesterday&period=day&format=xml&filter\_limit=10`](http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&date=yesterday&period=day&format=xml&filter_limit=10)

## XML Output

Here is the output of this request:

```
<?xml version="1.0" encoding="utf-8" ?>
<result>
```

```
<row>
    <label>Keyword not defined</label>
    <nb_uniq_visitors>50</nb_uniq_visitors>
    <nb_visits>50</nb_visits>
    <nb_actions>57</nb_actions>
    <max_actions>3</max_actions>
    <sum_visit_length>126</sum_visit_length>
    <bounce_count>44</bounce_count>
    <nb_visitsConverted>0</nb_visitsConverted>
    <idsubdatatable>47</idsubdatatable>
</row>
<row>
    <label>virtual drums</label>
    <nb_uniq_visitors>9</nb_uniq_visitors>
    <nb_visits>9</nb_visits>
    <nb_actions>10</nb_actions>
    <max_actions>2</max_actions>
    <sum_visit_length>5</sum_visit_length>
    <bounce_count>8</bounce_count>
    <nb_visitsConverted>0</nb_visitsConverted>
    <idsubdatatable>46</idsubdatatable>
</row>
<row>
    <label>virtual drum</label>
    <nb_uniq_visitors>2</nb_uniq_visitors>
    <nb_visits>2</nb_visits>
    <nb_actions>2</nb_actions>
    <max_actions>1</max_actions>
    <sum_visit_length>0</sum_visit_length>
    <bounce_count>2</bounce_count>
```

```
<nb_visitsConverted>0</nb_visitsConverted>
<idsubdatatable>48</idsubdatatable>
</row>
<row>
    <label>3d drums</label>
    <nbUniqVisitors>1</nbUniqVisitors>
    <nbVisits>1</nbVisits>
    <nbActions>1</nbActions>
    <maxActions>1</maxActions>
    <sumVisitLength>0</sumVisitLength>
    <bounceCount>1</bounceCount>
    <nbVisitsConverted>0</nbVisitsConverted>
    <idsubdatatable>43</idsubdatatable>
</row>
<row>
    <label>drums 3d</label>
    <nbUniqVisitors>1</nbUniqVisitors>
    <nbVisits>1</nbVisits>
    <nbActions>1</nbActions>
    <maxActions>1</maxActions>
    <sumVisitLength>0</sumVisitLength>
    <bounceCount>1</bounceCount>
    <nbVisitsConverted>0</nbVisitsConverted>
    <idsubdatatable>49</idsubdatatable>
</row>
<row>
    <label>drums virtual</label>
    <nbUniqVisitors>1</nbUniqVisitors>
    <nbVisits>1</nbVisits>
    <nbActions>1</nbActions>
```

```
<max_actions>1</max_actions>
<sum_visit_length>0</sum_visit_length>
<bounce_count>1</bounce_count>
<nb_visitsConverted>0</nb_visitsConverted>
<idsubdatatable>44</idsubdatatable>

</row>
<row>
    <label>virtual drumkit</label>
    <nb_uniq_visitors>1</nb_uniq_visitors>
    <nb_visits>1</nb_visits>
    <nb_actions>1</nb_actions>
    <max_actions>1</max_actions>
    <sum_visit_length>0</sum_visit_length>
    <bounce_count>1</bounce_count>
    <nb_visitsConverted>0</nb_visitsConverted>
    <idsubdatatable>50</idsubdatatable>

</row>
<row>
    <label>virtual drumming</label>
    <nb_uniq_visitors>1</nb_uniq_visitors>
    <nb_visits>1</nb_visits>
    <nb_actions>1</nb_actions>
    <max_actions>1</max_actions>
    <sum_visit_length>0</sum_visit_length>
    <bounce_count>1</bounce_count>
    <nb_visitsConverted>0</nb_visitsConverted>
    <idsubdatatable>45</idsubdatatable>

</row>
<row>
    <label>virtual drum set</label>
```

```

<nb_uniq_visitors>1</nb_uniq_visitors>
<nb_visits>1</nb_visits>
<nb_actions>1</nb_actions>
<max_actions>1</max_actions>
<sum_visit_length>0</sum_visit_length>
<bounce_count>1</bounce_count>
<nb_visitsConverted>0</nb_visitsConverted>
<idsubdatatable>42</idsubdatatable>
</row>
<row>
<label>virtual drumset musiciansfriends</label>
<nb_uniq_visitors>1</nb_uniq_visitors>
<nb_visits>1</nb_visits>
<nb_actions>1</nb_actions>
<max_actions>1</max_actions>
<sum_visit_length>0</sum_visit_length>
<bounce_count>1</bounce_count>
<nb_visitsConverted>0</nb_visitsConverted>
<idsubdatatable>51</idsubdatatable>
</row>
</result>

```

## Other useful examples

- XML of the visits of the last 10 days, one entry per day <http://demo.piwik.org/?module=API&method=VisitsSummary.getVisits&idSite=3&period=day&date=last10&format=xml>
- XML containing keywords from the last 3 weeks, one entry per week <http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&period=week&date=last3&format=xml>
- XML containing the keywords from the last 3 days which match the pattern "piwik"[http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&period=day&date=last3&format=xml&filter\\_column=label&filter\\_pattern=piwik](http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&period=day&date=last3&format=xml&filter_column=label&filter_pattern=piwik)

- RSS feed containing the top 30 keywords for the last 3 weeks, ordered by the number of actions people did when coming from these keywords [http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&period=week&date=last3&format=rss&filter\\_limit=30&filter\\_sort\\_column=3](http://demo.piwik.org/?module=API&method=Referers.getKeywords&idSite=3&period=week&date=last3&format=rss&filter_limit=30&filter_sort_column=3) You can get the data in one of these formats: XML, JSON, HTML, CSV, TSV, etc. See the [API Reference](#) for the documentation.

There are also functions for Websites, Users, Goals, PDF Reports (create, update, delete operations) and a lot more, such as: adding Annotations, creating custom Segments,

Check out the [Piwik API Reference](#)

# All About Analytics Data

---

## About this guide

### Read this guide if

- you'd like to know **how to aggregate, store and serve new analytics data for your plugin**
- you'd like to know **what the Archiving Process is and how it is used to automatically aggregate and cache analytics data**
- you'd like to know **how analytics data is stored and manipulated in PHP**
- you'd like to know **what segments are and how you can define your own**

### Guide assumptions

This guide assumes that you:

- can code in PHP,
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

## About Analytics

To analyze data is to search for patterns in a set of **things**. In Piwik those **things** are visits, web actions and goal conversions.

We search for patterns by **reducing** the set of things. Or in other words, we search for patterns by grouping individual things together to create subsets that are both recognizable and meaningful.

In Piwik the result of that grouping is the analytics data that it stores, displays and serves through an API. Read on to learn exactly what this data is, how Piwik calculates and stores it and how it is made available to Piwik users.

# Analytics Reports & Metrics

Piwik aggregates and persists two types of analytics data: **reports** and **metrics**. The difference between the two is that a **metric** is a single numeric value whereas a **report** is a two-dimensional array of values. **Reports** will normally contain **metrics**, but they can also contain other data.

## Core metrics

All reports are defined by plugins. Metrics can also be defined by plugins, but there are several, called **core metrics** that are defined and calculated by **Piwik Core**.

The following is a list of core metrics that relate to a set of visits:

- **Visits:** Number of tracked visits (a visit is series of events each of which happened no more than 30 minutes apart). \_Internally stored with the

```
'nb_visits'
```

metric name.

- **Unique Visitors:** The number of unique sources of visits (a visit source is an entity that causes a visit to be tracked). \_Internally stored with the

```
'nb_uniq_visitors'
```

metric name.

- **Actions:** The number of tracked actions (an action is an event tracked by Piwik). \_Internally stored with the

```
'nb_actions'
```

metric name.

- **Max Actions:** The maximum number of actions that occurred in one visit. \_Internally stored with the

```
'max_actions'
```

metric name.

- **Sum Visit Length:** The sum of each visit's elapsed time. *Internally stored with the*

```
'sum_visit_length'
```

metric name.

- **Bounce Count:** The number of visits that consisted of only one action. *Internally stored with the*

```
'bounce_count'
```

metric name.

- **Converted Visits:** The number of visits that caused at least one conversion. *Includes conversions for every goal of a site. Internally stored with the*

```
'nb_visits_converted'
```

metric name.

- **Conversions:** The number of conversions tracked for this set of visits. *Includes conversions for every goal of a site. Internally stored with the*

```
'nb_conversions'
```

metric name.

- **Revenue:** The total revenue generated by these visits. *Includes revenue for every goal of a site plus its ecommerce revenue. Internally stored with the*

```
'revenue'
```

metric name.

The following is a list of core metrics that relate to a single action type:

- **Hits:** The number times this action was ever done. *Internally stored with the*

```
'nb_hits'
```

*metric name.*

- **Sum Time Spent:** The total amount of time the user spent doing this action. *Internally stored with the*

`'sum_time_spent'`

metric name.

- **Sum Page Generation Time:** The total amount of time a server spent serving this action.*Internally stored with the*

`'sum_time_generation'`

metric name.

- **Hits With Generation Time:** The number of hits that included generation time information.*Internally stored with the*

`'nb_hits_with_time_generation'`

metric name.

- **Min Page Generation Time:** The minimum amount of time a server spent serving this action.*Internally stored with the*

`'min_time_generation'`

metric name.

- **Max Page Generation Time:** The maximum amount of time a server spent serving this action. *Internally stored with the*

`'max_time_generation'`

metric name.

- **Unique Exit Visitors:** The number of unique visitors that ever exited a site after this action.*Internally stored with the*

`'exit_nb_uniq_visitors'`

metric name.

- **Exit Visits:** The total number of visits that ended with this action. *Internally stored with the*

`'exit_nb_visits'`

metric name.

- **Unique Entry Visitors:** The total number of unique visitors that started a visit with this action. *Internally stored with the*

```
'entry_nb_uniq_visitors'
```

*metric name.*

- **Entry Visits:** The total number of visits that started with this action. *Internally stored with the*

```
'entry_nb_visits'
```

*metric name.*

- **Entry Actions:** ??? TODO: isn't this the same as entry visits? *Internally stored with the*

```
'entry_nb_actions'
```

*metric name.*

- **Entry Sum Visit Length:** The sum of each entry visit's elapsed time. *Internally stored with the*

```
'entry_sum_visit_length'
```

*metric name.*

- **Entry Bounce Count:** The number of visits that consisted of this action and no other. *Internally stored with the*

```
'entry_bounce_count'
```

*metric name.*

- **Hits From Search:** The number of times this action was done after a site search. *Internally stored with the*

```
'nb_hits_following_search'
```

*metric name.*

The following is a list of core metrics that relate to the set of ecommerce conversions (either all orders or all abandoned carts) recorded for a set of visits:

- **Revenue Subtotal:** The total cost of every item that was a part of these orders or abandoned carts. *Internally stored with the*

`'revenue_subtotal'`

*metric name.*

- **Revenue Tax:** The total tax amount applied to these orders/abandoned carts. *Internally stored with the*

`'revenue_tax'`

*metric name.*

- **Revenue Shipping:** The total amount of shipping applied to these orders/abandoned carts. *Internally stored with the*

`'revenue_shipping'`

*metric name.*

- **Revenue Discount:** The total amount of discounts applied to these orders/abandoned carts. *Internally stored with the*

`'revenue_discount'`

*metric name.*

- **Ecommerce Item Count:** The total number of items in these orders/abandoned carts. *Internally stored with the*

`'items'`

*metric name.*

TODO: necessary to document sum\_dialy\_nb\_uniq\_visitors (also sum\_daily\_exit\_nb\_uniq\_visitors & sum\_daily\_entry\_nb\_uniq\_visitors)? seems like a useless metric.

## Goal specific metrics

The following is a list of core metrics that relate to a set of visits and one goal of a site:

- **Goal Conversions:** The conversions tracked for a specific goal and this set of visits. *Stored in reports with a metric name of the format*

```
'goal_%idGoal%_nb_conversions'
```

- **Goal Revenue:** The total revenue generated by the conversions for a specific goal. *Stored in reports with a metric name of the format*

```
'goal_%idGoal%_revenue'
```

*Note: In the metric names displayed above,*

```
'%idGoal%'
```

*should be replaced with the ID of the goal in question.*

Goal specific metrics are stored in the database in the

```
'goals'
```

column of reports. The column contains a PHP array mapping goals with arrays of goal specific metric values. These values are set as column values with the metric names described above by the [AddColumnsProcessedMetricsGoalDataTable filter](#).

## Processed metrics

In the interests of efficiency (in terms of both the speed of the [Archiving Process](#) and the size of the database), many metrics are not stored in the database. These metrics can be calculated using other metrics and thus can be calculated right before reports are served. These metrics are collectively called **processed metrics**. Below is the list of processed metrics that are calculated using **core metrics**.

*Note: Some processed metrics will appear multiple times in the lists below. These metrics have different meanings based on the reports they are in.*

The following is a list of processed metrics that relate to a set of visits:

- **Conversion Rate:** The percent of visits that had at least one conversion. *Stored in reports with the*

```
'conversion_rate'
```

*metric name.*

- **Actions Per Visit:** The average number of actions for a single visit. *Stored in reports with the*

```
'nb_actions_per_visit'
```

*metric name.*

- **Average Time On Site:** The average number of time spent per visit in seconds. *Stored in reports with the*

`'avg_time_on_site'`

*metric name.*

- **Bounce Rate:** The percent of visits that resulted in a bounce. *Stored in reports with the*

`'bounce_rate'`

*metric name.*

The following is a list of processed metrics that relate to a single action type:

- **Average Generation Time:** The average amount of time it took for a server to serve this action. *Stored in reports with the*

`'avg_time_generation'`

*metric name.*

- **Average Number of Search Result Pages Viewed:** The average number of search result pages viewed after a site search. Only valid for site search keywords and site search categories. *Stored in reports with the*

`'nb_pages_per_search'`

*metric name.*

- **Average Time On Page:** The average amount of time users spent doing this action. *Stored in reports with the*

`'avg_time_on_page'`

*metric name.*

- **Entry Bounce Rate:** The percent of all visits that consisted of this action and no other. *Stored in reports with the*

`'bounce_rate'`

*metric name.*

- **Exit Rate:** The percent of all visits that ended with this action. *Stored in reports with the*

`'exit_rate'`

*metric name.*

The following is a list of processed metrics that relate to the set of ecommerce orders recorded for a set of visits:

- **Average Order Revenue:** The average revenue of each order. *Stored in reports with the*

`'avg_order_revenue'`

*metric name.*

The following is a list of processed metrics that relate to the set of ecommerce items in a set of orders or abandoned carts:

- **Average Price:** The average price of each item. *Stored in reports with the*

`'avg_price'`

*metric name.*

- **Average Quantity:** The average number of each item in an order/abandoned cart. *Stored in reports with the*

`'avg_quantity'`

*metric name.*

- **Product Conversion Rate:** The percent of orders/abandoned carts that include this item. *Stored in reports with the*

`'conversion_rate'`

*metric name.*

## Goal specific metrics

The following is a list of processed metrics that are also specific to one goal of one site:

- **Average Revenue per Visit:** The average amount of revenue generated per visit for this goal. *Stored in reports with the*

```
'goal_%idGoal%_revenue_per_visit'
```

*metric name.*

*Note: In the metric names displayed above,*

```
'%idGoal%'
```

*should be replaced with the ID of the goal in question.*

## Naming metrics

Plugins that want to calculate and persist their own metrics must give them a name with the following format:

```
"PluginName_metricName"
```

where **PluginName** is the name of the plugin and **metricName** is the name of the metric. For example:

```
"MyPlugin_myFancyMetric"
```

This naming convention is required in order to determine which plugins defines which metrics. Not following this convention will result in errors during the [Archiving Process](#).

**Core metrics** all have special names and do not follow this convention.

## Reports and DataTables

Reports are stored in memory using the [DataTable](#) class. A [DataTable](#) is an array of rows where each row is an array of columns.

Each row contains metrics that relate to a set of visits, actions, conversions or some other entity. The set is defined and described by a special **label** column. How the column describes the set depends entirely upon the specific report. For example, in the report returned by the [UserSettings.getBrowser](#) report a row with the label **Firefox** would hold metrics for the set of all visits that used the Firefox browser.

Some reports will not have a label column. These reports will have only one row that refers to the entire set of entities.

### Row metadata

In addition to metrics, each row can also contain metadata. This metadata will usually assist the label column in describing the set of things the row represents.

Some metadata have special meanings. For example, in much of Piwik metadata with the name

```
'logo'
```

is treated as a path to an image that is used to describe the row. This image is displayed alongside rows when reports are displayed in the UI.

Metadata with the name

'url'

is treated as a URL that describes the row and the label of the row is linked to this URL when reports are displayed in the UI.

### Subtables

Reports can be hierarchical. Each row in a report can be attached to another table of data. Any row in those tables can be attached to more tables, and so on ad infinitum. Tables that are attached to rows are called **subtables**.

Subtables provide further analytics for the set of visits that a row represents. For example, the **Actions.getPageUrls** report contains rows that describes a set of page view actions based on the first part of the page's URL. If this part is a directory and not a file, the row may have a subtable that describes that row's set of page view actions based on the second part of the page's URL.

Another example: the **Referrers.getSearchEngines** report contains a row for each search engine that was used in a visit. Each row will have a subtable that describes the keywords that were used with that search engine. The subtable rows will contain metric values for visits that used a specific keyword (determined by the subtable row) with a specific search engine (determined by the parent row).

## Naming Reports

Reports should be named in the same way as non-core metrics. That is, they should have a name with the following format:

"**PluginName\_reportName**"

where **PluginName** is the name of the plugin and **reportName** is the name of the report. For example:

"**MyPlugin\_myFancyReport**"

Plugins that do not follow this convention will cause errors during the [Archiving Process](#).

## Analytics Parameters

Reports and metrics provide analytics data about a set of things. Piwik determines what is in this set by using three constraints: a website ID, a period and a segment.

The website ID selects visits that were tracked for a specific website. This ID is specified in all HTTP requests by the **idSite** query parameter.

The period selects visits that were tracked within a specific date range. The period is specified in all HTTP requests by the **date** and **period** query parameters.

The segment is a condition that selects visits based on a boolean expression that uses visit properties. It is specified in all HTTP requests by the **segment** query parameter and can be used to select almost any subset of visits conceivable.

Analytics parameters are normally stored as report metadata (that is, they are stored as [DataTable](#)metadata).

**Every report and metric describes a set of things determined by these three parameters: the website, period and segment.**

## Report & Metric Persistence (Archive Data)

When persisted, reports and metrics are collectively termed **Archive Data**, which simply means that the data has been cached and does not need to be re-calculated.

Persisted reports and metrics are indexed by the website ID, period and segment. The date and time that the data was calculated and cached is also attached to each report and metric. *To learn the specifics of how this is done with MySQL see our guide [Persistence and the MySQL Backend](#).*

### Metric persistence

Metrics are numeric values and so there is nothing special done when persisting them. The website ID, period, segment and datetime of caching are attached to the metric value, and all this information is saved.

### Report persistence

Reports are complex data structures and so there is some extra processing required before they are persisted. The report's list of rows (an array of [DataTable\Row](#) instances) is serialized using PHP's [serialize](#) function. The string result is then compressed using [gzcompress](#). Finally, the website ID, period, segment and datetime of caching are attached to the compressed data, and all of this information is then saved.

#### Records

When a report is archived, it is called a **record** not a report. We make a distinction because multiple reports can sometimes be generated from one **record**.

For example, the UserSettings plugin uses one record to hold visits by browser information. This record is used to generate both the **UserSettings.getBrowserVersion** report and the **UserSettings.getBrowser** report. The second report simply processes the first in a way to make a new report. The plugin could have archived both reports, but this would have been a **massive** waste of space, considering the new report would be cached for every website/period/segment combination.

### Record storage guidelines

Care must be taken to store as little as possible when persisting records. Make sure to follow the guidelines below before inserting records as archive data:

- **Records should not be stored with string column names.** Instead they should be replaced with integer column IDs (see [Metrics](#) for a list of existing ones).
- **Metadata that can be added using existing data should not be stored with reports.** Instead they should be added in API methods when turning records into reports.

## The Archiving Process

Analytics data is calculated and cached on-demand. When a report for a specific website, period and segment (if any) is requested, Piwik will check if the data has been cached, and if not Piwik will generate and cache it.

Archiving logic (the logic that calculates and caches analytics data) is defined by individual plugins. When archiving is initiated, every report defined by a plugin is archived together, rather than individually.

If no segment is supplied in the data query and data cannot be found, every report of every plugin will be generated and cached all at once. If a segment is supplied, then the reports that belong to the same plugins as the requested data will be generated and cached.

### Plugin Archiver

Plugins that want to archive reports and metrics define a class called **Archiver** that extends from [Piwik\Plugin\Archiver](#). This class will be automatically detected and instantiated by Piwik during the archiving process.

## Report & Metric Aggregation

Reports and metrics are calculated differently based on the period type.

For day periods, the visits/actions/conversions/etc. (called **log data**) are themselves aggregated.

For other periods, the reports & metrics for the days within the periods are aggregated together. For example, when generating a report for a week period, the report for each day within the week (ie, Monday, Tuesday, Wednesday, etc.) will be queried and then aggregated together. This is far faster than aggregating each individual visit/action/etc. that was tracked during the entire week, but creates the same result. [1](#)

Log data aggregation is handled by the [LogAggregator](#) class. Archive data aggregation is handled by the [ArchiveProcessor::aggregateDataTableRecords](#) and [ArchiveProcessor::aggregateNumericMetrics](#) methods. Plugins can access a [LogAggregator](#) instance and a [ArchiveProcessor](#) instance through the [Piwik\Plugin\Archiver](#) class.

To learn more about how aggregation is accomplished with Piwik's MySQL backend, read our [Persistence and the MySQL Backend](#) guide.

[1] Because of this technique, we cannot calculate unique visitors for non-day periods.

## Report & Metric Caching

Reports and metrics are persisted using the [ArchiveProcessor](#) class. Metrics are inserted using the [ArchiveProcessor::insertNumericRecord](#) method. Reports are first serialized using the [DataTable::getSerialized](#) method and then inserted using the [ArchiveProcessor::insertBlobRecords](#) method:

```
$archiveProcessor = // ...

// insert a numeric value
$myFancyMetric = // ... calculate the metric value ...
$archiveProcessor->insertNumericRecord( 'MyPlugin_myFancyMetric',
    $myFancyMetric);

// insert a record (with all of its subtables)
$maxRowsInTable = Config::getInstance()->General[ 'datatable_archiving_maximum_rows_standard' ];

$dataTable = // ... build by aggregating visits ...
$serializedData = $dataTable->getSerialized($maxRowsInTable, $maxRowsInSubtable = $maxRowsInTable,
```

```
$columnToSortBy = Metrics::INDEX_NB_VISITS);

$archiveProcessor->insertBlobRecords('MyPlugin_myFancyReport',
$serializedData);
```

## Pre-archiving with the cron script

Though data is generated on demand, it would be highly inefficient and create a poor user experience if we relied on it for all users. Any user that receives a significant amount of visits would experience a large delay before being able to view their reports. Piwik solves this problem with a cron script that pre-archives data.

The cron script (called

`archive.php`

and located in

`/path/to/piwik/root/misc/cron`

) pre-archives data for every website and for every period except range periods. Reports & metrics for [stored segments](#) will also be archived.

The cron script will remember when it was last executed and will only initiate the archiving process for a website if there have been visits since that time.

## Disabling browser initiated archiving

For users that have websites that receive a lot of visits, simply allowing on-demand archiving through the browser will cause undesirable delays. These users can disable browser initiated archiving. [Read the user docs for more info.](#)

## Serving Reports

Reports are served through [APIs](#). API methods access persisted [records](#) transform them into presentable reports and serve them through Piwik's [Reporting API](#) either in HTTP responses or to PHP code (such as [Controller](#) methods).

## Transforming Records into Reports

As stated above, records are not the same as reports. Records are structured primarily to be stored not be read by either humans or other software. Thus API methods cannot simply access persisted data and return it. They must manipulated and made presentable.

[DataTable Filters](#)

[DataTable](#) instances, which are used to hold reports, are manipulated by either iterating through rows and manually making changes or through the use of [DataTable Filters](#). [DataTable Filters](#) manipulate [DataTable](#) instances in some way. There are several predefined ones that allow you to do common things easily.

Making a report presentable involves undo-ing the [changes that made it more efficient to store](#). Column names can be changed from integer IDs to string metric names via the [ReplaceColumnNames DataTable](#) filter:

```
$dataTable->filter('ReplaceColumnNames');
```

Metadata and [processed metrics](#) should also be added within API methods. [Existing filters](#) can be used to perform most of these tasks.

## API processing of reports

When a report is returned from an API method it goes through some extra processing based on what query parameters are set for the request. To see exactly what happens to a report, read the relevant section in our [Piwik's HTTP API](#) guide.

## Learn more

- To learn how log data and archive data are stored and processed in [MySQL](#) read our [Persistence and the MySQL Backend](#) guide.
- To learn more about how reports are served in the [Reporting API](#) read our [Piwik's HTTP API](#) guide.
- To learn more about how reports are displayed read our [Visualizing Report Data](#) guide.

# All About Tracking

---

## About this guide

### Read this guide if

- you'd like to know how to use the [HTTP tracking API](#)
- you'd like to know how [plugins can extend the tracker and track new data](#)
- you'd like to know the [tracking system extracts information from tracking requests](#)
- you'd like to know how the [tracker inserts data into log tables](#)

## Guide assumptions

This guide assumes that you:

- can code in PHP,
- are knowledgeable about how HTTP works (eg, request & response headers),
- have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide),

# Tracker Functionality

The Piwik Tracker tracks the data that Piwik analyzes. Tracker clients send HTTP requests to Piwik and based on the values of certain query parameters and HTTP request fields, visits, actions and conversions are tracked. This document explains exactly how this process works.

## Types of Tracking Requests

Different types of tracking requests will do different things. There are three types of tracking requests that the Piwik Tracker will recognize. These are requests to track visitor actions, requests to manually convert a goal and requests to track ecommerce orders. These three actions cannot be done simultaneously.

### Visit Tracking Request

This type of tracking request is one that tracks visit related information such as a pageview, outlink or download. When the tracker receives this type of request, it will do the following things:

1. checks if this visit is from a returning visitor
2. if this is not a returning visitor, a new visit is tracked
3. if this is a returning visitor, the tracker examines the last visit action of the visitor. If the action currently being tracked occurred over 30 minutes after the last known action of the visitor, a new visit is created. If not, the ongoing visit is queried.
4. the visit action is recorded

### Visitor Detection

Returning visitors are detected by checking if the visitor ID sent with the tracking request is a known visitor ID or if the visit configuration used by the visitor has been seen before.

The visitor ID is set by the tracking client and stored as a cookie. When the tracker finds a visitor ID in the database that matches the one in the cookie, we know there is a returning visit. We don't, however, use the visitor ID alone as it is not always valid. Some browsers

(or just browsers that are configured a certain way) will create new visitor IDs on each pageview. This is why we also try to match a visit's configuration.

The configuration of a visit includes:

- the operating system used,
- the browser used,
- the browser's version,
- the visitor's IP address,
- the language used in the browser
- and the browser plugins enabled.

If a visitor ID does not match (or there is no visitor ID) and the tracker sees a configuration that is exactly the same as the current visit's, it will assume the new visit is from that returning visitor.

The use of visit configuration does create a risk of attributing a visit to the wrong visitor, but we believe this inaccuracy is far better than the creation of a new visit on every pageview.

### Action Type Creation

Visit actions are not recorded with the URLs and page titles of the actions. Instead, the URLs and page titles are stored in [Action Type](#) entities and visit actions link to them by ID.

Action types are created when visit actions with new URLs, page titles or other action data are found.

### Geolocation

The location of a visitor is provided by an implementation of the [LocationProvider](#) base type. The implementation to use is stored in the **usercountry.location\_provider** option.

Currently, Piwik provides two ways of geolocating browser visits. The default method guesses the country by the browser language used. The other method uses a [GeoIP](#) database to geolocate visits using the visitor's IP address.

New LocationProviders can be created simply by subclassing the [LocationProvider](#) class and making sure the file is included with the rest of a plugin. Piwik will automatically know it is available.

### Conversion Tracking Request

Conversion tracking requests are tracking requests that have the

`idGoal`

query parameter set. These requests do just one thing: they trigger a conversion for a particular

goal.

TODO: looks like visits and actions are recorded too?

## Ecommerce Tracking

TODO:

## The Tracker Cache

The tracker mostly queries and manipulates the log tables, but sometimes it needs to use other data. For example, options that tell the tracker what visits to exclude and which location provider to use when geolocating. The queries used to get this data are not expensive, but doing them on every tracking request would result in performance degradation.

So instead, the tracker caches this information in a file that is read on every request. When users change this data in the database, the relevant part of Piwik will invalidate the cache so on the next tracking request the tracker will re-query and cache the data.

### Adding more to the tracker cache

Plugins can add data to the tracker cache by handling the [Tracker.setTrackerCacheGeneral](#) event. If a plugin needs to add data that is associated with a specific site, it can add the data through the [Site.getSiteAttributes](#) event.

When data that should be in the tracker cache is changed or removed, the [Cache::deleteTrackerCache](#) method should be called so the next tracking request will query and cache the new data.

## Scheduled Tasks

Tracking requests can also execute scheduled tasks. This is done so scheduled tasks will still execute even for users who don't setup the [archive.php cron script](#).

Scheduled tasks will only execute if the tracker is not authenticated, if the tracker was not executed through the command line and if the

`[Tracker] record_statistics`

INI config option is set to

`1`

. They are always executed after the tracking request is handled.

TODO: authenticated to any user or just super user?

## Debugging the Tracker

To verify that your data is being tracked properly, you can enable debug logging in the Piwik tracking file, **piwik.php**.

[Read more.](#)

## Extending the Tracker

Plugins can add extra data to tracked visits by handling the [Tracker.newVisitorInformation](#) event. The UserCountry plugin uses this event to add geolocation information to a visit and the DevicesDetection plugin uses this event to add device information beyond the operating system used.

Plugins that want to add new data to the log tables themselves can alter the tables to add new columns. See [this section in the Persistence & the MySQL Backend guide](#).

## The Tracking HTTP API

To track page views, events, visits, you have to send a HTTP request to your Tracking HTTP API endpoint, for example, **http://your-piwik-domain.tld/piwik.php** with the correct query parameters set.

[View the Tracking HTTP API Reference docs.](#)

## Learn more

- For a **list of tracking clients** see this [page](#).
- To learn more **about geolocation** read the [GeolP user docs](#).
- To learn **about Piwik's JavaScript tracker** read our [documentation for the tracker](#).

## MVC (Model-View-Controller) in Piwik

---

## About this guide

**Read this guide if**

- you'd like to know **how HTTP requests are handled by Piwik**

- you'd like to know what a **Controller or API** in Piwik is
- you'd like to know how to show **HTML generated by your plugin** to Piwik users
- you'd like to know how to use **Twig templates** in Piwik

## Guide assumptions

This guide assumes that you:

- can code in PHP,
- have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide),
- and have knowledge of the [Model-View-Controller pattern](#).

# HTTP Request Handling

Piwik's MVC (Model-View-Controller) implementation is the first bit of code that is executed when Piwik handles an HTTP request.

Every request that is sent to Piwik's reporting side (as opposed to Piwik's tracking side) is sent to the index.php file in Piwik's root directory. This file creates an instance of the [FrontController](#) and uses it to dispatch the current request.

[FrontController](#) looks for the **module** and **action** query parameters and invokes the controller method specified by **action** in the plugin specified by **module**. If no **action** is specified, Piwik sends the request to the controller's

**index**

method.

The controller method that is called has to do one thing, which is return something that can be

**echo**

'd. As a plugin developer you can do this in any way you'd like, Piwik won't stop you, but the convention used by the rest of Piwik is to create a Piwik [View](#), query APIs to request any needed data and then render the view. For example:

```
class Controller extends \Piwik\Plugin\Controller
{
    public function index()
    {
        $view = new View("@MyPlugin\\index.twig");
        $view->data = \Piwik\Plugins\MyPlugin\API::getInstance()
            ->getData();
```

```
    return $view->render();  
}  
}
```

If a plugin with the above code was enabled and loaded, the request `?module=MyPlugin&action=index` would invoke the

```
index()  
method above. ?module=MyPluginwould have the same effect.
```

Read on to learn more about the individual components in this workflow.

## Piwik APIs (Models)

Piwik APIs serve two purposes: they serve the data used in views and they [automatically expose plugin functionality through a HTTP API](#).

In this guide, we discuss the first purpose.

### About API Classes

Plugins provide APIs by defining a class named API that derives from [Piwik\Plugin\API](#). Every public method that does not have the

```
@ignore  
annotation is exposed as part of Piwik's Reporting HTTP API.
```

All APIs are singletons. To access API methods programatically the [Singleton::getInstance](#) method must be called first:

```
MyAPI::getInstance()->doSomething();
```

#### API methods

API methods can take any number of parameters. Since they can be called through HTTP, methods must assume that parameters will be passed as strings. This also means that method parameters can only be simple values, such as

```
string  
s,  
bool  
s,  
numeric  
s, etc.
```

API methods can return only one of four types of data: either a simple value (

```
string
```

```
,  
    bool  
,
```

`numeric`, etc), a [DataTable](#) instance, a [DataTable\Map](#) instance or an array of any of these values. This is so Piwik will always be able to convert the result into the desired output format in the reporting API.

If an API method encounters an error, it should throw an exception. Piwik will be able to convert exceptions to the desired output format in the reporting API.

### API method security

All API methods should check whether the current user is allowed to invoke the method. If the API method is read-only, this means checking that the user has view access to the resources the method returns. If the API method does something, this normally means checking that the user has admin access to the functionality (or alternatively checking that the user is the super user). For example,

```
public function getAllForSite($idSite)  
{  
    Piwik::checkUserHasViewAccess($idSite);  
  
    return // ...  
}
```

Look at the

```
check...  
methods in the Piwik class to see what types of checks can be made.
```

### Calling API methods

API methods can be called in two ways. They can be called directly after getting the API singleton instance:

```
MyAPI::getInstance()->doSomething(  
    Common::getRequestVar('idSite'),  
    Common::getRequestVar('date'),  
    Common::getRequestVar('period')  
);
```

or they can be called using the [Piwik\API\Request](#) class:

```
Request::processRequest("MyAPI.doSomething");
```

Note how in the second method, the [Common::getRequestVar](#) method (which safely retrieves query parameter values) does not have to be called.

The [Piwik\API\Request](#) class will forward the current request parameters to the API method which makes using it the better choice in some situations.

Also note, that when [Piwik\API\Request](#) is used, [extra processing is applied to report data](#).

## Piwik Views (Views)

Piwik Views are classes that implement [ViewInterface](#). The main view class [Piwik\View](#) will use a [Twig](#)template that is specified upon construction to generate output. There is also another class called[ViewDataTable](#) that is used specifically to visualize analytics data.

Using a view is straightforward. First, it is configured. The meaning of this is different based on the View type. For [Piwik\View](#) instances, it simply means setting properties, for example:

```
$view = new View("@MyPlugin/myTemplate.twig");  
  
// set properties  
$view->property1 = 'property1';  
$view->property2 = 'here's another property';
```

For [ViewDataTable](#), it's a bit more complicated.

Once a view is configured, it is rendered via the [ViewInterface::render](#) method:

```
return $view->render();
```

This is the same for all view types.

## Twig Templates

The preferred way to generate anything text-based (like HTML) using data is to define Twig templates and use [View](#). Plugin developers should not accomplish this task with new view types unless they need to output something that is not text-based (such as an image).

*If you do not know how to create Twig templates, learn how by [reading Twig's documentation](#).*

### Template storage and referencing

Templates are stored in the **templates** subdirectory of a plugin's root directory. When you create a [View](#) instance you must tell it what template it should use using a string with the following format: `"**PluginName**/**TemplateFileName**". Piwik will look for a file named **TemplateFileName** in the **PluginName** plugin's **templates** subdirectory.

Template files in Piwik have a very specific naming convention. If the file contains the output for a specific controller method, the file should be named after the method. For example, **myControllerMethod.twig**. In all other cases, the file should be named after what it contains and be prefixed with an underscore. For example, **\_myEmbeddedWidget.twig**.

### Twig functions and filters

The [View](#) class adds several filters and functions before rendering a template. It will also define properties that the template can use. To learn exactly what's defined, read the [View class docs](#).

## Piwik Controllers

Controllers are the objects in Piwik that output HTML. Every plugin that wants to output HTML should define its own Controller that extends [Piwik\Plugin\Controller](#).

Every public method in a controller is exposed and can be called through an HTTP request. **When creating your controller, care should be taken to avoid exposing methods that don't need to be. It may be possible for an attacker to use these methods.**

## Controller Output

Controller methods should

**return**

ing their output (as opposed to

**echo**

ing it). Piwik will assume the output is HTML and will automatically take care of the appropriate HTTP response headers. If you want to output something other than HTML you will have to output the **Content-Type** HTTP response header. For example:

```
@header('Content-Type: application/json; charset=utf-8');
```

## Using Controller Methods in the Piwik UI

Adding a controller method to a plugin's controller will allow it to be executed via an HTTP request, but it won't automatically show it in the Piwik UI somewhere. There are two ways to make the result of a controller method appear in the Piwik UI:

- add a new menu item that links to the controller method
- use AJAX to invoke the controller method and then display the result

Here's how you do both:

### Adding controller methods as menu items

Menu items are added through an event. Each menu has its own event:

- [Menu.Admin.addItem](#) for the menu shown on the left when a user clicks *Settings*.
- [Menu.Top.addItem](#) for the menu shown at the very top of each page.
- [Menu.Reporting.addItem](#) for the main menu shown on non-admin pages (the menu [just below the logo](#)) TODO: link to image here

Plugins can call the [MenuAbstract::add](#) method within event handlers for these events to add menu items. For example, the following will add a menu item titled **My Menu Item** to the admin menu that links to the **MyPlugin.myPage** controller method.

```
// an event handler for Menu.Admin.addItem

public function addAdminMenuItems()
{
    MenuAdmin::add(
        "My Menu Category",
        "My Menu Item",
        array('module' => 'MyPlugin', 'action' => 'myPage'),
        $isVisible = true,
        $order = 10
    );
}
```

### Invoking controller methods via AJAX

If you have your own custom JavaScript running on Piwik, you can use AJAX to dynamically invoke controller methods and display the result. For example:

```
// invoke MyPlugin.myPage and append the result to the end of the #root element

var ajax = new ajaxHelper();
ajax.addParams({
    module: 'MyPlugin',
```

```

action: 'myPage'
}, 'get');

ajax.setCallback(function (response) {
    $('#root').append(response);
});

ajax.setFormat('html');

ajax.send(false);

```

## Controller method conventions

### Getting query parameters

Unlike API methods, controller methods do not take query parameters as input. If you need to access a query parameter value, use the [Common::getRequestVar](#) method.

**To avoid XSS vulnerabilities, never access**

```

$_GET
/
$_POST
directly, always go throughCommon::getRequestVar.

```

### Generating Output

As a plugin developer you are welcome to generate your output in any way you'd like (as long as it's secure), there is nothing in Piwik that will force you to code a certain way. That being said, most Piwik controller methods will have the following convention:

```

public function myControllerAction()
{
    // Step 1: if this controller action is supposed to execute
    some logic, do that first

    $idSite = Common::getRequestVar('idSite');
    $period = Common::getRequestVar('period');

    $somethingResult = MyDoer::doSomething($idSite, $period);

    // Step 2: create a view to render the output

    $view = new View("@MyPlugin/myControllerAction.twig");
}

```

```

    // Step 3: set properties of the view, getting data from API
    //s when necessary

    $view->somethingResult = $somethingResult;

    $view->neededData = API::getInstance()->getNeededData();

    // Step 4: render the view

    return $view->render();

}

```

## Calling API methods

Since controller methods do not take query parameter values as method parameters it can sometimes be a pain to invoke API methods within controller methods. In this case, controllers make use of the [Piwik\API\Request](#) class which will forward all query parameters to an API method. For example, let's look at some of the code in the **save** method in the [Annotations](#) plugin controller:

```

$view = new View('@Annotations/saveAnnotation');

// NOTE: permissions checked in API method

// save the annotation

$view->annotation = Request::processRequest("Annotations.save"
"');

return $view->render();

```

The code invokes the Annotations API's **save** method forwarding all query parameters so the controller method doesn't have to call [Common::getRequestVar](#) several times.

## Reusing Controller methods

Sometimes you may want to use a controller method that belongs to another controller (to say embed a control provided by another controller). You can use the [FrontController::fetchDispatch](#) method to accomplish this:

```

// controller method in our plugin's controller

public function index()
{

```

```
$view = new View("@MyPlugin/index.twig");

$view->realtimeMap = FrontController::getInstance()->fetchDispatch($module = "UserCountryMap", $method = "realtimeMap");

return $view->render();

}
```

### Checking for correct HTTP methods

To maintain correct HTTP semantics, some controller methods should check whether the correct HTTP request method was used to invoke them. For example, tasks are normally executed via a **POST** rather than a **GET**. Controller methods that handle these tasks should check whether a POST was used:

```
public function myAdminTask()

{
    // ... do some stuff ...

    if ($_SERVER["REQUEST_METHOD"] != "POST") {
        return;
    }

    // ... do some stuff ...
}
```

## Controller Security

Like API methods, controller methods should make sure the current user is both valid and authorized to perform the requested action or view the requested data. This means calling the [access checking methods](#) that should also be called in API methods and checking that the supplied **token\_auth** is valid (via [Controller::checkTokenInUrl](#)).

Here's an example of a secure controller method:

```
public function myAdminTask()

{
    Piwik::checkUserIsSuperUser();

    $this->checkTokenInUrl();
```

```
if ($_SERVER["REQUEST_METHOD"] != "POST") {  
    return;  
}  
  
// ... do some stuff ...  
}
```

## Learn more

- To learn **how to display analytics data in a controller method** read our [Visualizing Report Data](#) guide.
- To learn **how Piwik automatically exposes API classes in a HTTP API** read our [Piwik's HTTP API](#) guide.
- To learn **more about writing secure code with Piwik** read our [Security](#) guide.
- To learn **how to add custom JavaScript in addition to your plugin's HTML output** or**how to style your plugin's HTML output** read our [Working with Piwik's UI](#) guide.

# Piwik's Reporting API

---

## About this guide

### Read this guide if

- you'd like to know **how Piwik's Reporting API works and how your plugin can extend it**
- you'd like to know **what Piwik's Reporting API does to data before it is served**
- you'd like to know **what output formats can be used to view data that is returned from the Reporting API**
- you'd like to know **about query parameters that are used by the Reporting API to transform analytics data**

### Guide assumptions

This guide assumes that you:

- can code in PHP,

- understand how reports are stored in memory (if not, read this section of our [All About Analytics Data](#) guide),
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

# The Reporting API

Piwik's **Reporting API** allows third party applications to access analytics data and manipulate miscellaneous data (anything other than reports or tracked data) through HTTP requests.

## API Requests

Every HTTP request to the **API.index** controller method will be handled by the Reporting API. Valid requests **must** have a query method named **method** that references the API method to invoke, for example, **UserSettings.getBrowser**.

API requests are processed in the following way:

1. The **API.index** controller method creates a [Piwik\API\Request](#) instance and uses it to dispatch the API request.
2. The [Piwik\API\Request](#) instance uses the **method** query parameter to determine which plugin and API method to use.
3. The [Piwik\API\Request](#) instance invokes the plugin's API method using query parameters as arguments. Query parameters with the same name as method parameters will be passed as those method parameters. For example, **idSite** query parameter will be passed as the

```
$idSite
```

method parameter.

4. The [Piwik\API\Request](#) instance uses a [ResponseBuilder](#) object to process the result of the API method and convert it into the desired output format.
5. The output is returned to the **API.index** controller method and is eventually

```
echo
```

'd.

## API methods

The Reporting API invokes methods that are found in each plugin's [API class](#). Every public method in these classes is included as part of the Reporting API except for those whose documentations contain the `@ignore` annotation.

Query parameters are passed as method parameters, so API methods must assume method parameters will be string or array values. Objects are not allowed as parameters.

Methods are only allowed to return **numeric** values, **string** values, **arrays**, [DataTable](#) instances or[DataTableMap](#) instances. Piwik will not know how to format anything else.

**Note:** When returning [DataTable](#) or [DataTableMap](#) instances, filters will need to be applied. Make sure to queue filters that are used for presentation purposes.

If a method throws an exception, its message will appear in the output. The stack trace can be displayed during debugging by changing `ResponseBuilder::DISPLAY_BACKTRACE_DEBUG` to true.

To see the list of API methods your Piwik install supports, click the **API** link at the very top of every Piwik page. See the demo's list [here](#).

## Extra report processing

Reports that are returned by API methods, either in the form of a [DataTable](#) instance or[DataTable\Map](#) instance, are manipulated before they are outputted. A set of [DataTable\Filter](#)s are executed on the reports based on the values of certain query parameters.

The following is a list of filters that are applied, what they do and what query parameters control whether they will run or not. The order in which they appear in this list is also the order in which they are applied:

1. **Flattener:** This filter will merge an entire [DataTable](#) hierarchy into one [DataTable](#), adding rows of subtables to their parent rows. It will only be applied if the `flat` query parameter is set to

1

2. **Pattern:** Removes rows of a [DataTable](#) that do not match a regex pattern. Will be applied if the `filter_pattern` query parameter is set to a regex. The `filter_column` query parameter dictates which column to apply the regex pattern to (defaults to `label`).
3. **PatternRecursive:** Removes rows of a [DataTable](#) for which the row and all subtables of the row do not match a regex pattern. Will be applied if

the **filter\_pattern\_recursive** query parameter is set to a regex.

The **filter\_column\_recursive** query parameter dictates which column to apply the regex pattern to (defaults to **label**).

4. **[ExcludeLowPopulation](#)**: Deletes all rows that contain a specific column whose value is lower than a minimum threshold value. Will be applied if the **filter\_excludelowpop** query parameter is set. It should be set to the column to check. The **filter\_excludelowpop\_value** query parameter specifies the minimum threshold value (defaults to

```
0
```

).

5. **[AddColumnsProcessedMetrics](#)**: Adds some universal processed metrics to each row of the report. Will be applied if the **filter\_add\_columns\_when\_show\_all\_columns** query parameter is set to

```
1
```

6. **[AddColumnsProcessedMetricsGoal](#)**: Adds processed metrics to each row for each goal of a site. Will be applied if the **filter\_update\_columns\_when\_show\_all\_goals** query parameter is set to

```
1
```

. The **idGoal** query parameter controls which goal to use. It can be the ID of a goal or one of these special values:

- **[AddColumnsProcessedMetricsGoal::GOALS\\_OVERVIEW](#)**: if used, the filter will add metrics for the goals overview and not individual goals.
- **[AddColumnsProcessedMetricsGoal::GOALS\\_MINIMAL\\_REPORT](#)**: if used, no per-goal metrics will be added, just one metric, **revenue\_per\_visit**.
- **[AddColumnsProcessedMetricsGoal::GOALS\\_FULL\\_TABLE](#)**: if used, will display per-goal metrics for every goal of the site including the ecommerce goal.

**idGoal** defaults to **[AddColumnsProcessedMetricsGoal::GOALS\\_OVERVIEW](#)**.

1. **[Sort](#)**: Sorts a **DataTable**. Will be applied if the **filter\_sort\_column** query parameter is set to the column that should be sorted by. The **filter\_sort\_order** query parameter determines in what order the table is sorted. Can be either

```
'desc'
```

or

'asc'

2. **Truncate**: Removes all rows after a certain row index. Will be applied if the **filter\_truncate** query parameter is present. The parameter should be set to the row number after which rows should be removed.
3. **Limit**: Removes rows not within a certain row index range. Will be applied if the **filter\_limit** query parameter is supplied and set to an integer. This is the size of the range. The **filter\_offset** query parameter determines the starting row index of the range. The **keep\_summary\_row** query parameter will make sure the summary row stays in the report if the parameter is set to

1

4. **SafeDecodeLabel**: Urldecodes and then sanitizes **label** column values. This filter is always applied.
5. **Queued Filters**: All of a [DataTable](#)'s queued filters are applied at this point. They will not be applied if the **disable\_queued\_filters** query parameter is present and set to

1

6. **ColumnDelete**: Removes columns from every row based on a query parameter that lists what to delete or a query parameter that lists what to keep. Will be applied if either the **hideColumns** query parameter or the **showColumns** query parameter are defined. Both should be a comma separated list of column names.
7. **LabelFilter**: This filter will remove all rows except the one (or ones) specified by the **label** query parameter. Only applied if the **label** query parameter is set. The **label** query parameter can be a single value or a path to a row in a subtable. To descend into subtables, the value should contain the

>

character, for example,

```
urldir>urlsubdir>index
```

. **label** can also be an array of values, for example,

```
label[]>arg1&label[]>arg2
```

## Other special query parameters

There are some other special query parameters that affect the way reports are processed:

- **disable\_generic\_filters**: If set to

```
1
```

, the filters numbered 2 - 9 above will not be applied. Defaults to

```
0
```

so if this parameter is absent, the filters will be applied.

- **format**: Determines the [output format](#) of the return value. This affects all return values regardless of whether it is a report (ie, stored within a [DataTable](#) or [DataTable\Map](#) or not).

## Output formats

The output of a Reporting API request is a serialized string of the API method's return value. The format of this string is determined by the value of the **format** query parameter. Currently Piwik supports the following output formats:

- [xml](#)
- [json](#)
- [csv](#)
- [tsv](#) (Excel)
- [html](#) (A simple HTML representation, does not use [report visualizations](#).)
- [php](#) (A serialized PHP array.)

There is a special output format value, **original**, that can be used when requesting data from within Piwik using [Piwik\API\Request](#). This format will force the result to be returned as unprocessed and unserialize data.

*Note: The [Request::processRequest](#) method automatically uses the **original** format, so in most cases you won't need to specify*

`format=original`

## Special API Methods

Some of the API methods in the API plugin have special meaning and uses:

### Report Metadata

The **API.getMetadata**, **API.getReportMetadata** and **API.getProcessedReport** API methods can be used to get information about one or all reports. The information includes the metrics contained in the report, documentation for those metrics and [more](#).

These methods can be used by third party applications that provide an interface to the analytics stored by Piwik. **API.getReportMetadata**, which returns metadata for every report, can be used to get a list of all available reports for a website. **API.getMetadata** can be used to get more information about a single report. **API.getProcessedReport** can be used to get the metadata of a single report along with the report's data.

Report metadata can also be used within Piwik Core or within plugins for features that operate on a report or reports specified by the user. For example, the **ImageGraph** plugin, which outputs an image of a graph using report data, uses report metadata values as hints for how to draw the output graph. The **ScheduledReports** plugin also uses report metadata in a similar way.

TODO: is there an official set of properties for report metadata?

### Row Evolution

Piwik's [row evolution feature](#) that is available through the UI is also available through the Reporting API. Third party applications can use the **API.getRowEvolution** method to get both [single row evolution data](#) or [multi-row evolution data](#).

### Bulk API Requests (API.getBulkRequest)

[Like the Tracking API](#), the Reporting API supports bulk requests. A bulk request allows applications to invoke and retrieve the results for multiple API methods with one HTTP request. This can save time and processing resources for most requests.

To send a bulk request, send an HTTP request to the **API.getBulkRequest** API method. The only required query parameter is named **urls**. It should be an array of individual API request URLs. For example:

```
http://demo.piwik.org/?module=API&method=API.getBulkRequest&format=xml&urls[]module%3DAP%26method%3DVisitorInterest.getNumberOfVisitsPerVisitDuration%26format%3DXML%26idSite%3D7%26period%3Dday%26date%3D2013-11-24%26expanded%3D1&urls[]module%3DAP%26method%3DUserSettings.getBrowser%26format%3DXML%26idSite%3D7%26period%3Dday%26date%3D2013-11-24%26expanded%3D1
```

This example uses the following API requests:

- module=API&method=UserSettings.getBrowser&format=XML&idSite=7&period=day&date=2013-11-24&expanded=1
- module=API&method=VisitorInterest.getNumberOfVisitsPerVisitDuration&format=XML&idSite=7&period=day&date=2013-11-24&expanded=1

*Note: The separate API methods are executed synchronously, so for long running API methods, using a bulk request may be a bad idea.*

## Other Methods

- **API.get**: Calls the

```
get
```

API method of all loaded plugins that support it and merges the result. The

```
get
```

methods all output the metrics that the plugin archives, so the result of **API.get** is values for every metric your Piwik install supports (for the specified website & period).

- **API.getSegmentDimensionMetadata**: Returns metadata for every supported [segment dimension](#). The following information is returned for each segment dimension:

- **'type'**

: The type of segment dimension.

- **'category'**

: A translated string that describes the segment dimension's category.

- o **'name'**

: A translated string or a translation token that describes the segment dimension itself.

- o **'segment'**

: The segment dimension's ID. This is what gets used in segment expressions.

- o **'acceptedValues'**

: A string that describes what values should be used with the segment dimension.

- o **'sqlSegment'**

: The table column the segment dimension operates on, for example,

**'log\_visit.idvisitor'**

- o **'sqlFilterValue'**

: An optional PHP callback that transforms the value supplied in a segment expression so it can be used in an SQL expression.

- o **'permission'**

: Whether the current user can use this segment dimension.

- **API.getSuggestedValuesForSegment**: Returns a list of values that can be used with a specified segment dimension.
- **API.getLogoUrl**: Returns a URL to the Piwik logo.
- **API.getHeaderLogoUrl**: Returns a URL to a smaller version of the Piwik logo.

## Learn more

- To learn how API classes are used internally read our [MVC in Piwik](#) guide.
- To learn about how to calculate a report read our guide [ALI About Analytics Data](#).

# Querying the Reporting API

---

This guide explains how to call the Piwik API to request your web analytics data. There are two methods:

- using the standard HTTP API over HTTP
- using the Piwik PHP library files directly

## Call the Piwik API using the HTTP API over HTTP

If you want to request data in any language (PHP, Python, Ruby, ASP, C++, Java, etc.) you can use the HTTP API. It is a simple way to request data via standard HTTP GET.

**Security Notice:** if the API call requires the token\_auth and the HTTP request is sent over untrusted networks, we highly advise that you use an encrypted request. Otherwise, your token\_auth is exposed to eavesdroppers. This can be done using https instead of http. In the following example, replace the string "http" by "https".

You can, for example, get your top 100 search engine keywords used to find your website during the current week. Here is an example in PHP:

```
<?php  
exit; // REMOVE this line to run the script  
  
// this token is used to authenticate your API request.  
// You can get the token on the API page inside your Piwik interface  
$token_auth = 'anonymous';  
  
// we call the REST API and request the 100 first keywords for the last month for the idsite=1  
$url = "http://demo.piwik.org/";  
$url .= "?module=API&method=Referrers.getKeywords";  
$url .= "&idSite=7&period=month&date=yesterday";  
$url .= "&format=PHP&filter_limit=20";
```

```

$url .= "&token_auth=$token_auth";

$fetched = file_get_contents($url);
$content = unserialize($fetched);

// case error
if (!$content) {
    print("Error, content fetched = " . $fetched);
}

print("<h1>Keywords for the last month</h1>");
foreach ($content as $row) {
    $keyword = htmlspecialchars(html_entity_decode(urldecode($row['label'])), ENT_QUOTES, ENT_QUOTES);
    $hits = $row['nb_visits'];

    print("<b>$keyword</b> ($hits hits)<br>");
}

```

Here is the output of this code:

```

<h1>Keywords for the last month</h1><b>Keyword not defined</b>
(14834 hits)<br><b>all website</b> (7 hits)<br><b>downloads anz
eigen</b> (5 hits)<br><b>fake driving licence id cards</b> (4 h
its)<br><b>piwik open page overlay</b> (3 hits)<br><b>piwik vis
itor log empty</b> (3 hits)<br><b>premature end of data</b> (3
hits)<br><b>show all websites</b> (3 hits)<br><b>=&#1079;&#1080;
&#1084;&#1085;&#1080; &#1087;&#1072;&#1083;&#1090;&#1072;</b>
(3 hits)<br><b>@get_loaded_extensions()</b> (2 hits)<br><b>acce
ss control</b> (2 hits)<br><b>all websites</b> (2 hits)<br><b>a
pache &ccedil;ant:access:directory</b> (2 hits)<br><b>cannot us
e output buffering in output buffering display handlers</b> (2
hits)<br><b>change piwik server</b> (2 hits)<br><b>child pid 14
600 exit signal segmentation fault</b> (2 hits)<br><b>cyrus fa
ke driveng license</b> (2 hits)<br><b>faux permis de conduire</

```

```
b> (2 hits)<br><b>faux permis de conduire italien</b> (2 hits)<br><b>faux permis de conduire suisse</b> (2 hits)<br>
```

## Call the Piwik API in PHP

If you want to request data in a PHP script that is on the same server as Piwik, you can use this simple technique. This is a more efficient solution as it doesn't require network calls. You directly call the PHP Piwik runtime and get the PHP data structure back.

If you are developing a plugin, you have to use this technique.

```
<?php

use Piwik\API\Request;
use Piwik\FrontController;

define('PIWIK_INCLUDE_PATH', realpath('../..'));
define('PIWIK_USER_PATH', realpath('../..'));
define('PIWIK_ENABLE_DISPATCH', false);
define('PIWIK_ENABLE_ERROR_HANDLER', false);
define('PIWIK_ENABLE_SESSION_START', false);

// if you prefer not to include 'index.php', you must also define here PIWIK_DOCUMENT_ROOT
// and include "libs/upgradephp/upgrade.php" and "core/Loader.php"

require_once PIWIK_INCLUDE_PATH . "/index.php";
require_once PIWIK_INCLUDE_PATH . "/core/API/Request.php";

FrontController::getInstance()->init();

// This inits the API Request with the specified parameters
$request = new Request(
    method=UserSettings.getResolution
    &idSite=7
)
```

```

        &date=yesterday
        &period=week
        &format=XML
        &filter_limit=3
        &token_auth=anonymous
');

// Calls the API and fetch XML data back
$result = $request->process();
echo $result;

```

Here is the output of this script:

```

<?xml version="1.0" encoding="utf-8" ?>
<result>
    <row>
        <label>1920x1080</label>
        <nb_visits>1668</nb_visits>
        <nb_actions>4912</nb_actions>
        <max_actions>85</max_actions>
        <sum_visit_length>397261</sum_visit_length>
        <bounce_count>1008</bounce_count>
        <nb_visitsConverted>51</nb_visitsConverted>
        <sum_daily_nb_uniq_visitors>1390</sum_daily_nb_uniq_visitors>
    </row>
    <row>
        <label>1366x768</label>
        <nb_visits>978</nb_visits>
        <nb_actions>2318</nb_actions>
        <max_actions>40</max_actions>
        <sum_visit_length>175275</sum_visit_length>
    </row>

```

```
<bounce_count>673</bounce_count>
<nb_visitsConverted>40</nb_visitsConverted>
<sum_daily_nb_uniq_visitors>863</sum_daily_nb_uniq_visitors>
</row>
<row>
    <label>1680x1050</label>
    <nb_visits>663</nb_visits>
    <nb_actions>2259</nb_actions>
    <max_actions>277</max_actions>
    <sum_visit_length>158866</sum_visit_length>
    <bounce_count>373</bounce_count>
    <nb_visitsConverted>17</nb_visitsConverted>
    <sum_daily_nb_uniq_visitors>576</sum_daily_nb_uniq_visitors>
</row>
</result>
```

# Theming

---

## About this guide

In Piwik, **themes** are special types of plugins that change the look and feel of Piwik's UI. They use CSS and LESS to override the default styles defined by other Piwik plugins. This guide will explain how to create a new theme.

### Read this guide if

- you'd like to know [how to create your own themes for Piwik](#)

### Guide assumptions

This guide assumes that you:

- can code in PHP and JavaScript,

- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

## Creating a theme

To create an empty theme, run the following command from Piwik's root directory:

```
./console generate:theme
```

After you enter the appropriate information, a theme will be created for you in the **plugins** directory.

## Adding styles

Every theme has one main file that contains all of your theme's styling overrides. The location of this file is determined by the **stylesheet** property in your theme's **plugin.json** file. The property's value is a path relative to the plugin's root directory:

```
{
  "name": "MyTheme",
  "description": "A new theme.",
  "theme": true,
  "stylesheet": "stylesheets/theme.less"
}
```

The generated theme will already have a file for your new styles, so you don't need to set this property. The file is called **theme.less** and is located in the **stylesheets** directory of your theme.

You can put your entire theme into this one file if you want, but this will not result in easy to read and easy to maintain code. Instead, you should group your theme's styles based on the part of Piwik they modify and place them in separate LESS files. In **theme.less** you can [@import](#) them.

## Adding JavaScript files

Themes can also add new JavaScript files. These files can be used to style things that can't be styled through CSS or LESS, such as radio buttons.

To add JavaScript files, add them as an array to the **javascript** property in your theme's **plugin.json** file:

```
{  
    "name": "MyTheme",  
    "description": "A new theme.",  
    "theme": true,  
    "stylesheet": "stylesheets/theme.less",  
    "javascript": ["javascripts/myJavaScriptFile.js", "javascripts/myOtherJavaScriptFile.js"]  
}
```

## Theming colors

Colors that are used in CSS are simple to override. Simply set the colors for the element, for example:

```
// changing the dataTable header column color  
table.dataTable th {  
    color: red;  
}
```

## Theming colors used in JavaScript & PHP

Some colors are only used in JavaScript and in PHP. We've made it possible for those colors to be specified through CSS, but the process is a bit different than setting colors of normal HTML elements.

Each color used in JavaScript is given a name and grouped in a *color namespace*. You can set these colors like this:

```
.color-namespace-name[data-name=color-name] {  
    color: red;  
}
```

For example,

```
.bar-graph-colors[data-name=grid-background] {  
    color: @background-color-base;  
}  
  
.bar-graph-colors[data-name=grid-border] {
```

```
color: @basic-grid-border-color;  
}
```

## Named colors

Here is a list of all named colors in Piwik:

- *Namespace:* **sparkline-colors** — contains colors for sparkline images.
  - **backgroundColor:** The background color of sparkline images.
  - **lineColor:** The color of the line in the sparkline.
  - **minPointColor:** The color of the point that marks the minimum value observed in the data set.
  - **lastPointColor:** The color of the point that marks the last value of the data set.
  - **maxPointColor:** The color of the point that marks the maximum value observed in the data set.
- *Namespace:* **bar-graph-colors:** contains colors for bar graph [report visualizations](#).
  - **grid-background:** The background color of the graph.
  - **series1:** The color of the bars representing the first series of data.
  - **series2:** The color of the bars representing the second series of data.
  - **series3:** The color of the bars representing the third series of data.
  - **series4:** The color of the bars representing the fourth series of data.
  - **series5:** The color of the bars representing the fifth series of data.
  - **series6:** The color of the bars representing the sixth series of data.
  - **series7:** The color of the bars representing the seventh series of data.
  - **series8:** The color of the bars representing the eighth series of data.
  - **series9:** The color of the bars representing the ninth series of data.
  - **series10:** The color of the bars representing the tenth series of data.
  - **ticks:** The color of x-axis gridlines and x-axis ticks.
  - **single-metric-label:** The color of the series name label if only **one** series is displayed. If you don't care about whether there's one series displayed or not, set this color to the one you used in **series1**.
- *Namespace:* **pie-graph-colors:** contains colors for pie graph [report visualizations](#).
  - **grid-background:** The background color of the graph.
  - **series1:** The color of the pie graph segment representing the first value in the data displayed.
  - **series2:** The color of the pie graph segment representing the second value in the data displayed.
  - **series3:** The color of the pie graph segment representing the third value in the data displayed.

- **series4**: The color of the pie graph segment representing the fourth value in the data displayed.
  - **series5**: The color of the pie graph segment representing the fifth value in the data displayed.
  - **series6**: The color of the pie graph segment representing the sixth value in the data displayed.
  - **series7**: The color of the pie graph segment representing the seventh value in the data displayed.
  - **series8**: The color of the pie graph segment representing the eighth value in the data displayed.
  - **series9**: The color of the pie graph segment representing the ninth value in the data displayed.
  - **series10**: The color of the pie graph segment representing the tenth value in the data displayed.
  - **single-metric-label**: The color of the series name label if only **one** series is displayed. If you don't care about whether there's one series displayed or not, set this color to the one you used in **series1**.
- *Namespace*: **evolution-graph-colors**: contains colors for evolution graph [report visualizations](#) (the big line graphs that display data over time).
  - **grid-background**: The background color of the graph.
  - **series1**: The color of the line representing the first series of data displayed.
  - **series2**: The color of the line representing the second series of data displayed.
  - **series3**: The color of the line representing the third series of data displayed.
  - **series4**: The color of the line representing the fourth series of data displayed.
  - **series5**: The color of the line representing the fifth series of data displayed.
  - **series6**: The color of the line representing the sixth series of data displayed.
  - **series7**: The color of the line representing the seventh series of data displayed.
  - **series8**: The color of the line representing the eighth series of data displayed.
  - **series9**: The color of the line representing the ninth series of data displayed.
  - **series10**: The color of the line representing the tenth series of data displayed.
  - **ticks**: The color of x-axis gridlines and x-axis ticks.

- **single-metric-label**: The color of the series name label if only **one** series is displayed. If you don't care about whether there's one series displayed or not, set this color to the one you used in **series1**.
- **Namespace: realtime-map**: contains colors for the [Realtime Visitors Map](#).
  - **white-bg**: The background color for the map when using the *light theme*.
  - **white-fill**: The country/region fill color for the map when using the *light theme*.
  - **black-bg**: The background color for the map when using the *dark theme*.
  - **black-fill**: The country/region fill color for the map when using the *dark theme*.
  - **visit-stroke**: The border color for each visit.
  - **website-referrer-color**: The fill color of a visit whose referrer was a website (*only used if the Referrer Type color mode is used*).
  - **direct-referrer-color**: The fill color of a visit that has no referrer (*only used if the Referrer Type color mode is used*).
  - **search-referrer-color**: The fill color of a visit whose referrer was a search engine(*only used if the Referrer Type color mode is used*).
  - **live-widget-highlight**: The color to use when highlighting a visit in the live widget (*only used when embedding both the realtime map and the [Live widget](#) in the dashboard*).
  - **live-widget-unhighlight**: The color to use when unhighlighting a visit in the live widget(*only used when embedding both the realtime map and the [Live widget](#) in the dashboard*).
  - **symbol-animate-fill**: The starting fill color to use when animating the appearance of a new visit. The ending fill color is always the visit's normal color.
- **Namespace: visitor-map**: contains colors for the [Visitors Map](#).
  - **no-data-color**: The fill color for countries that have no visits.
  - **one-country-color**: The fill color for countries that have visits, if only one country received visits.
  - **color-range-start-choropleth**: The start of the color range used to color countries and regions based on the number of visits.
  - **color-range-end-choropleth**: The end of the color range used to color countries and regions based on the number of visits.
  - **color-range-start-normal**: The start of the color range used to color cities based on the number of visits.
  - **color-range-end-normal**: The end of the color range used to color cities based on the number of visits.
  - **country-highlight-color**: The fill color to use when a country is *highlighted* (a country is highlighted when the mouse enters it while the shift key is pressed).
  - **country-selected-color**: The fill color to use when a country is *selected* (ie, when a country is being used in a row evolution popup).

- **unknown-region-fill-color**: The fill color to use for regions that have no visits (or 0 of whatever metric is displayed).
- **unknown-region-stroke-color**: The stroke color to use for regions that have no visits (or 0 of whatever metric is displayed).
- **region-stroke-color**: The stroke color for regions with visits.
- **region-selected-color**: The fill color to use when a region is *selected* (ie, when a region is being used in a row evolution popup).
- **region-highlight-color**: The fill color to use when a region is *highlighted* (a region is highlighted when the mouse enters it while the shift key is pressed). Only regions with visits can be highlighted.
- **invisible-region-background**: The fill color to use for regions when displaying cities. This color should make the city data more visible.
- **region-layer-stroke-color**: The stroke color of each region.
- **city-label-color**: The color of city labels.
- **city-stroke-color**: The stroke color to use for cities.
- **city-highlight-stroke-color**: The stroke color to use when a city is *highlighted* (a city is highlighted when the mouse enters it).
- **city-highlight-fill-color**: The fill color to use when a city is *highlighted* and the shift key is pressed (a city is highlighted when the mouse enters it).
- **city-highlight-label-color**: The label color to use while the shift key is pressed for a city that is *highlighted*.
- **city-label-fill-color**: The fill color for a city label. <!-- TODO: the city label fill color is never set initially only when unhighlighting. Will cause bugs for themes.)
- **city-selected-color**: The fill color to use for a city when it is *selected* (ie, when a city is being used in a row evolution popup).
- **city-selected-label-color**: The label color to use for a city when it is *selected* (ie, when a city is being used in a row evolution popup).
- **special-metrics-color-scale-1**: The start of the color scale used to color countries and regions for *special metrics*. [1]
- **special-metrics-color-scale-2**: The second color in the color scale used to color countries and regions for *special metrics*.
- **special-metrics-color-scale-3**: The third color in the color scale used to color countries and regions for *special metrics*.
- **special-metrics-color-scale-4**: The end of the color scale used to color countries and regions for *special metrics*.

[1] Metrics that use the *special metrics* color scale are the following: **avg\_time\_on\_site**, **nb\_actions\_per\_visit** and **bounce\_rate**

## colors.less

It is a common convention among themes to put all color values in a **colors.less** file as LESS variables, for example:

```
@lightBlack = #ccc;
```

```
@darkred = darken(red, 30%);  
// etc.
```

This file is the first file imported in the `theme.less` file.

## Making a plugin themable

Plugins that define their own UI widgets or new [report visualizations](#) are, for the most part, already themable. As long as they rely entirely on CSS for the look and feel, they can be easily themed.

If these new widgets or visualizations use colors in JavaScript or PHP, more work must be done to make them themable.

## Using CSS colors in JavaScript

To use colors defined in CSS within JavaScript, the [ColorManager](#) class must be used. Using it is straightforward. After you define some named colors in CSS like this:

```
.my-color-namespace[data-name=my-color-name] {  
    color: red;  
}  
  
.my-color-namespace[data-name=my-second-color] {  
    color: blue;  
}
```

You access them through [ColorManager](#) like this:

```
var ColorManager = require('piwik').ColorManager;  
  
// get one color  
  
var myColorToUse = ColorManager.getColor('my-color-namespace',  
    'my-color-name');  
  
// get multiple colors all at once  
  
var myColorsToUse = ColorManager.getColor('my-color-namespace',  
    ['my-first-color', 'my-second-color']);
```

# Using CSS colors in PHP

Using CSS colors in PHP is more complicated, in fact, so much so that **using color values in PHP is discouraged**. The only way to use them is to access them in JavaScript, and then pass them to PHP via query parameters:

```
// get the colors

var ColorManager = require('piwik').ColorManager;

var myColorsToUse = ColorManager.getColor('my-color-namespace',
  ['my-first-color', 'my-second-color']);


// set the source of an <img> to use those colors

var jsonColors = JSON.stringify(myColorsToUse);

$('img#myImage').attr('src', '?module=MyPlugin&action=generateImage&colors=' + encodeURIComponent(jsonColors))
```

Then use those colors in PHP:

```
// controller method in MyPlugin

public function generateImage()
{
  $colors = Common::getRequestVar('colors', null, $dataType =
  'json');

  $myFirstColor = $colors['my-first-color'];
  $mySecondColor = $colors['my-second-color'];

  // ... generate the image ...
}
```

## Learn more

- To learn more about creating new report visualizations read our [Visualizing Report Data](#) guide.
- To learn more about writing JavaScript for Piwik plugins and themes read our [Working with Piwik's UI](#) guide.

# Security in Piwik

---

## About this guide

**Read this guide if**

- you'd like to know **how to write secure code when extending Piwik or contributing to Piwik**

### Guide assumptions

This guide assumes that you:

- can code in PHP, JavaScript and SQL,
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

## Introduction

If you plan on developing a Piwik plugin or [contributing to Piwik Core](#) your code must be secure. You should make an effort to avoid vulnerabilities.

This guide contains a list of methods to combat certain vulnerabilities. Follow all of them when creating your plugin or contribution.

## Preventing XSS

[XSS](#) is the injection of malicious scripts into a webapp's UI. Either by storing malicious scripts in entity data (like website names for Piwik), attackers can gain control of applications that do not take the proper precautions.

### Always load request parameters

via [Common::getRequestVar](#)

In your PHP code, if you need access to a variable in

`$_GET`

or

`$_POST`

, always use [Common::getRequestVar](#) to get it. `getRequestVar` will sanitize the request variable so

if an attacker passes

```
<script>...</script>
```

your code will end up using

```
&lt;script&gt;...&lt;/script&gt;
```

. This will help you avoid accidentally embedding unescaped text in HTML output.

For text you know may contain special characters or if you need to output text in a format that doesn't need XML/HTML sanitization (like JSON), call the [Piwik::unsanitizeInputValues](#) to undo the sanitization.

*Note: You can sanitize text that isn't in a request variable by calling [Piwik::sanitizeInputValues](#).*

## Use |raw sparingly in Twig templates

When creating [Twig](#) templates, try to avoid using the

```
| raw
```

filter when possible. As an alternative, try putting the HTML you want to reuse in a separate template and

```
{% include %}
```

it.

If you do use

```
| raw
```

, make sure what your inserting has been properly escaped.

## Be careful when using jQuery.html

In your JavaScript, be careful when using the

```
$.html
```

method to insert HTML into the DOM. Make sure the string you are inserting came from Piwik and has been escaped.

If you know that the text your inserting shouldn't be HTML, then **do not use**

```
$.html
```

, instead use the

```
$.text
```

or

```
$.val
```

methods. For example:

```
var ajaxData = getDataFromAjax();
```

```
$( '#someLabel' ).text(ajaxData.labelToUse);
```

# Preventing CSRF

[CSRF](#) is a form of attack where an attacker gets a user to click a link on their website that does something the user would not want on the webapp the user uses. The link could, for example, point to a Piwik controller method that changes the user's password.

This attack can be prevented with one technique:

## Check for the `token_auth`

In every controller method you create that changes Piwik settings, changes a user's settings or does some other admin level function, call the [Controller::checkTokenInUrl](#) method. For example:

```
// method in a controller

public function doSomeAdminStuff()
{
    $this->checkTokenInUrl();

    // ...
}
```

In every API method that executes some admin level function, make sure to check for the proper user permissions by calling one of the [Piwik::check...](#) methods. For example:

```
// method in an API class

public function changeSettingsForUser($userLogin)
{
    Piwik::checkUserIsSuperUserOrTheUser($userLogin);
}
```

## `token_auth` in the browser

Your JavaScript should send the `token_auth` to controller and API methods that need it, but you should make sure the **token\_auth never appears in the URL**. This way, it will never be saved by the browser in any way.

To keep the `token_auth` out of a browser cache, plugins can use POST requests.

TODO: make sure Reporting API guide has security stuff

# Preventing SQL Injection

[SQL Injection](#) is the manipulation of an app's SQL by sending SQL in fields as parameters that are used to construct SQL statements.

For example, if an app builds an SQL statement like this:

```
$sql = "SELECT * from mytable where id = " . $_GET['id'];
```

, an attacker could pass

```
"1 OR 1"
```

for the **id** query parameter to cause the query to output every row in **mytable**.

SQL injection can be prevented by doing one thing:

## Use placeholders in your SQL

When writing SQL statements, use [SQL placeholders](#) instead of directly inserting variables into your statement. In other words, **don't do this**:

```
use Piwik\Db;

$idSite = Common::getRequestVar('idSite');

$sql = "SELECT * FROM " . Common::prefixTable('site') . " WHERE
        idsite = " . $idSite; // DON'T DO THIS!!

$rows = Db::query($sql);
```

Instead, **do this**:

```
use Piwik\Db;

$idSite = Common::getRequestVar('idSite');

$sql = "SELECT * FROM " . Common::prefixTable('site') . " WHERE
        idsite = ?";

$rows = Db::query($sql, array($idSite));
```

There is a limit to the number of placeholders you can use. If you need to use more placeholders than the limit allows, you may have to concatenate the parameters directly. Make sure these parameters are obtained from a trusted source (such as from another

query). This is done in `ArchiveSelector::getArchiveData` with archive IDs. Since the IDs are obtained from another query, it's safe to use them in this way.

## Preventing Remote File Inclusion

[Remote File Inclusion](#) is the inclusion and execution of source code that is not part of the webapp. It happens in PHP with

```
include
```

or

```
require
```

statements that use a path determined by the user.

In Piwik, the best way to prevent remote file inclusion attacks is to just never

```
require
```

/

```
include
```

files using data from the user. Instead, **put logic in classes that can be loaded by Piwik's autoloader** and instantiate/use different classes based on data obtained from the user. In other words, **don't do this**:

```
$clientToUse = Common::getRequestVar('seoClient');

require_once PIWIK_INCLUDE . '/plugins/MyPlugin/Clients/' . $clientToUse . '.php'; // DON'T DO THIS!!

$client = new $clientToUse();

// ... use $client ...
```

Instead, **do this**:

```
$clientToUse = Common::getRequestVar('seoClient');

if ($clientToUse == 'mySeoProvider') {
    $client = new Clients\MySeoProvider();
} else if ($clientToUse == 'myOtherSeoProvider') {
    $client = new Clients\MyOtherSeoProvider();
} else {
```

```
        throw new Exception("Invalid SEO provider client: $clientTo
Use!");
}

// ... use $client ...
```

## Preventing Direct Access

**Direct access** is simply the possibility of accessing one of your plugin's PHP files and having them execute. If some code does execute, it will display error messages that reveal valuable information to an attacker.

To prevent this type of vulnerability, put the following at the top of your PHP files that would execute something when run directly:

```
<?php

defined('PIWIK_INCLUDE_PATH') or die('Restricted access');
```

## Other Coding Guidelines

Here are some other coding guidelines that will help make your code more secure:

- **PHP files should start with a**

```
<?php
```

**tag that is never closed.**

- **Use the**

```
.php
```

**extension for all your PHP scripts.**

- **Avoid executing php code using one of the following functions: [eval](#), [exec](#), [passthru](#), [system](#), [popen](#) or [preg\\_replace](#) (with the**

```
"e"
```

**modifier).**

- Make sure that accessing your files directly doesn't execute any code that could have an impact on your Piwik install.

- Make sure your code doesn't rely on

`register_globals`

set to

`On`

. Note: PHP5 sets

`register_globals`

to

`Off`

by default.

- \*\*If your plugin has admin functionality (functionality only an administrator or the super user can use) then your [Controller](#) must extend [Piwik\Plugin\Controller\Admin](#).
- Some servers will disable PHP functions for (undisclosed) security reasons. Replacement functions can sometimes be found in `libs/upgradephp/upgrade.php`, including

`_parse_ini_file()`

,

`_glob()`

,

`_fnmatch()`

, and

`_readfile()`

. The functions

`safe_serialize()`

and

```
safe_unserialize()
```

are like the built-in functions, but won't serialize & unserialize objects. TODO: is this useful at all? for security or for something else?

TODO: what about: "Handle user/untrusted input & Handling output" both in plugins.md security section. don't know what it means.

## Learn more

- To learn **more about security in web applications** read this article: [Top 10 Security from The Open Web Application Security Project \(OWASP\)](#).
- To learn **more about security in PHP applications** read this two part article: [part 1](#), [part 2](#).

# Internationalization

---

## About this guide

### Read this guide if

- you'd like to know **how to make your plugin available in other languages**
- you'd like to know **how to make your contribution to Piwik Core available in other languages**

### Guide assumptions

This guide assumes that you:

- can code in PHP and JavaScript,
- can create Twig templates,
- know what [internationalization](#) is,
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

# The Basics

Internationalization is accomplished in Piwik by replacing text in the code with string IDs that describe the text. The string IDs are associated with actual text in languages that Piwik supports. These associations are stored in the JSON files in the

`lang`

directory.

The string IDs are called **translation tokens** and have the following format:

`MyPlugin_DescriptionOfThisText`

. When Piwik needs to replace a token with translated text, it will look at what language is currently selected and replace the token with the text from that language's JSON file. If no entry can be found, Piwik will default to the english text.

Translated text entries are allowed to contain

`sprintf`

parameters, for example,

`"This translated text is uses a %s parameter"`

or

`"This translated text %1$s uses %2$s parameters."`

. Every translate function will accept extra parameters that get passed to

`sprintf`

with the text.

## Using internationalization in PHP

To translate text in PHP, use the [Piwik::translate](#) function. For example,

```
$translatedText = Piwik::translate('MyPlugin_MyText');
```

or

```
$translatedText = Piwik::translate('MyPlugin_MyParagraphWithALink', '<a href="http://piwik.org">', '</a>');
```

## Using internationalization in Twig Templates

To translate text in Twig templates, use the

`translate`

filter. For example,

```
{{ 'MyPlugin_MyText' | translate }}
```

or

```
{{ 'MyPlugin_MyParagraphWithALink' | translate('<a href="http://piwik.org">', '</a>') }}
```

## Using internationalization in JavaScript

Translating text in the browser is a bit more complicated than on the server. The browser doesn't have access to the translations and we don't want to send every translation file to every user just so a couple lines of text can be translated.

Piwik solves this problem by allowing plugins to define which translation tokens should be available in the browser and sending only the translations of those keys in the current language to the browser.

To mark a translation token so it will be available client side use the [Translate.getClientSideTranslationKeys](#) event:

```
// an event handler in MyPlugin.php
public function getClientSideTranslationKeys(&$translationKeys)
{
    $translationKeys[] = 'MyPlugin_MyText';
    $translationKeys[] = 'CorePlugin_SomeCoreText';
}
```

To use these translations in JavaScript, use the global

```
_pk_translate
JavaScript function:
var translatedText = _pk_translate('MyPlugin_MyText');
```

## Adding translation tokens

If you are developing a plugin or theme add the translation token to your plugin's language file. TODO: where does the language file go?

If you are developing a contribution for Piwik Core add the translation token and the english translation to `lang/en.json`.

TODO: show example (after we figure)

## Guidelines for new translation tokens

Follow these guidelines when creating your own translation tokens:

1. **Reuse!** If a core plugin contains a translation you can use, use that instead. If there's a translation you want to use but can't because it's in the wrong case, try using functions like

```
lcfirst
```

and

```
ucfirst
```

2. **Reduce redundancy in your translated text.** If the same text appears in multiple translated text entries, try to move the translated text out by using sprintf parameters. For example, if you have text entries like:

```
"You cannot use commas."
```

and

```
"You cannot use underscores."
```

Try to split them up into something like:

```
"You cannot use %s."
```

```
"commas"
```

and

```
"underscores"
```

This guideline is more important for contributions to Piwik Core than for new plugins.

## Automated Tests

---

### About this guide

Read this guide if

- you'd like to know **how to add automated testing to your plugin so you can catch bugs before your users do**
- you'd like to know **how to run the testing suite used to test Piwik core**

## Guide assumptions

This guide assumes that you:

- can code in PHP,
- can use PHPUnit,
- have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide),
- and understand how Piwik handles requests (if not, read our [MVC in Piwik](#) guide).

# Piwik's automated testing suite

Piwik Core contains suite of tests used to make sure Piwik works properly and new commits do not introduce new bugs. There are three types of tests in this suite: **unit tests**, **integration tests** and **UI tests**.

**Unit tests** test individual classes to make sure their methods work properly. **Integration tests** test Piwik's [Reporting API](#) and [archiving logic](#) by tracking visits and checking that the output of certain API queries matches the expected output. **UI tests** tests Piwik's twig templates, JavaScript and CSS by tracking visits, generating screenshots of URLs with [phantomjs](#) and comparing expected screenshots with processed ones.

## UI tests

Unit and integration tests are fairly straightforward to run. UI tests, on the other hand, need a bit more work. To run UI tests you'll need to install [phantomjs version 1.9 or higher](#) and make sure

`phantomjs`

is on your PATH. Then you'll have to get the tests which are located in another repository but are included in Piwik as a submodule:

`git submodule init`

`git submodule update`

If you're on Ubuntu, you'll also need some extra packages to make sure screenshots will render correctly:

```
sudo apt-get install ttf-mscorefonts-installer imagemagick imagemagick-doc
```

## Running tests

Piwik Core's tests can be run in two ways. The first is to use the [console](#) command line tool by running:

```
./console test:run
```

or to run a specific set of test suites

```
./console test:run Core # for unit tests
```

```
./console test:run Integration
```

To run UI tests or the tests in a single file in the set of unit or integration tests, you will have to use the second method, which is to call [phpunit](#) directly:

```
# first, we must be in the PHPUnit directory, so
cd tests/PHPUnit
phpunit Core
phpunit Integration
phpunit UI
```

or

```
phpunit Core/CommonTest.php
phpunit Integration/ArchiveCronTest.php
phpunit UI/UIIntegrationTest.php
```

## Testing your plugins

If you're creating a new plugin that defines new reports or has some complex logic, you may find it beneficial to engage in [Test Driven Development](#) or at least to verify your code is correct with tests. With tests you'll be able to ensure that your code works and you'll be able to ensure the changes you make don't cause regressions.

At the moment, you can write unit or integration tests for your plugins. This section will explain how.

*Note: All test files must be put in a **tests** directory located in the root directory of your plugin and every test you write should have the*

```
@group
```

*set to the name of your plugin (for example,*

```
@group MyPlugin
```

).

## Writing unit tests

To create a simple unit test that doesn't need a MySQL database to test against, create a test case that extends [PHPUnit\\_Framework\\_TestCase](#). If your test will need access to a test Piwik database, create a test case that extends the **DatabaseTestCase** class.

**DatabaseTestCase** is a Piwik test class that provides **setUp** and **tearDown** logic that creates a test database with Piwik tables.

## Writing integration tests

To create an integration test, create a test case that extends the **IntegrationTestCase** base class. Then implement the **getApiForTesting**. This method should return an array of arrays. Each nested array contains information for a single test.

The first element in the array should be one or more API methods or the

```
'all'
```

string. This determines which API methods whose output should be compared against expected files. The second element should be an associative array that contains a set of options that affect the way the test is run or URL used to invoke the API method. You are allowed to set the following options:

- **testSuffix**: The suffix added to the output file name. If you call a single API method more than once in an integration test, all but one of them should have a **testSuffix** set so different output files will be created.
- **format**: The desired format of the output. Defaults to

```
'xml'
```

. The extension of the output is determined by the format.

- **idSite**: The ID of the website to get data for or

```
'all'
```

- **date**: The date to get data for.
- **periods**: The period or periods to get data for. Can be an array. For example,

```
'day'
```

or

```
array('day', 'mont')
```

- **setDateLastN**: Flag describing whether to query for a set of dates or not.
- **language**: The language to use.
- **segment**: The segment to use.
- **visitorId**: Sets the visitorId query parameter to this value.
- **abandonedCarts**: Sets the abandonedCarts query parameter to this value.
- **idGoal**: Sets the idGoal query parameter to this value.
- **apiModule**: The value to use in the apiModule request parameter.
- **apiAction**: The value to use in the apiAction request parameter.
- **otherRequestParameters**: An array of extra request parameters to use.
- **disableArchiving**: If true, disables archiving before running tests.

Some examples:

```
// test a single API method
array('UserSettings.getResolution', array('idSite' => $idSite,
'date' => $dateTime)),

// test all methods in a plugin
array('API', array('idSite' => $idSite, 'date' => $dateTime)),

// test every API method
array('all', array('idSite' => $idSite, 'date' => $dateTime)),

// set some custom request parameters
array('API.getBulkRequest', array('format' => 'xml',
'testSuffix' => '_bulk_xml',
'otherRequestParameters' => array(
'urls' => $bulkUrls))),

// test multiple dates w/ multiple periods and multiple sites
array('UserSettings.getResolution', array('idSite' => 'all',
```

```
'date' => $dateTime,  
      'periods' => array('day'  
, 'week', 'month', 'year'),  
      'setDateLastN' => true)),
```

After implementing `getApiForTesting`, add the following test to the file:

```
/**  
 * @dataProvider getApiForTesting  
 * @group Integration  
 */  
public function testApi($api, $params)  
{  
    $this->runApiTests($api, $params);  
}
```

This will test every API method specified in `getApiForTesting`.

## Fixtures

Before you can run your tests, you'll have to set the test's fixture. **Fixtures** add test data to the database by adding websites, tracking visits, etc.

To set a fixture, add a

```
public static  
field named  
$fixture
```

to your test class and initialize it below the class definition, for example:

```
namespace Piwik\Plugins\MyPlugin\Tests;
```

```
class MyIntegrationTest extends IntegrationTestCase  
{  
    public static $fixture = null;  
  
    // ...  
}
```

```
MyIntegrationTest::$fixture = new \Test_Piwik_Fixture_ThreeGoalsOnePageview();
```

To see the fixtures Piwik defines, see the files in the **tests/PHPUnit/Fixtures** directory.

TODO: list them here?

## Expected and processed output

Integration tests will generate an expected output file for every API method and period combination. The generated output (also called **processed** output) is stored in the **processed** subdirectory of your plugin's **tests** directory. The expected output should be stored in a directory named **expected**.

When you first create an integration test, there will be no expected files. You will have to copy processed files to the expected folder after ensuring they are correct.

## Writing UI tests

TODO: ensure they can be written

## Running plugin tests

To run the tests for your plugin, run the following command in the root of your Piwik install:

```
./console tests:run MyPlugin
```

Where

```
MyPlugin
```

should be replaced with the name of your plugin.

## Learn more

- To learn **more about what you can do with phpunit** read phpunit's [user docs](#).

# Manual UI Testing

# About this guide

This document contains a list of manual UI checks and tests that can be performed in order to ensure the UI works.

Piwik includes a [UI test suite](#) but these tests do not test everything. **If you have modified any of the Javascript, CSS, HTML, or PHP code that can affect the User Interface, you must go through this list to make sure everything still works.**

**Read this guide if**

- you've made a change to the core UI and need to make sure there are no regressions

## Asset Merging

Piwik comes with a dynamic JS/CSS/LESS file minifier and merger. Learn more about it here: [Blog post — Making Piwik UI Faster](#).

During development you may find it necessary to disable this system. **You should enable it before performing UI tests.** You should also make sure merged assets are created with your changes. To do this, disable or enable a plugin in Piwik.

*Learn how to enable/disable the asset merger in our [Working with Piwik's UI](#) guide.*

## Browsers to test with

Piwik should work with the following browsers:

- Firefox,
- Firefox with Firebug extension, check that there is no error or notice in the JS code,
- Internet Explorer 8+
- Internet Explorer with Debug enabled (set debugging in Tools > Internet > Advanced > Debug),
- Opera,
- Safari,
- Chrome

Try and run the following tests with as many of these as you can.

To help with your testing you may want to generate screenshots of Piwik pages using a tool like [Browser Shots](#).

# The Table Report Visualization

This is the UI widget that displays a report as a table with columns and rows.

Check that the following things are true:

- label column icons are present (eg, icon of google/yahoo/..., flags for countries, etc.).
- the style (background color) is different for odd & even rows.
- the label text is truncated when it is too long and when it is truncated there is a tooltip that displays the original text.
- column documentation displays when hovering over a column name in the HTML table view.

And test the following features:

- clicking a column heading to sort by column (the *column sorted* icon should appear over the sorted column once the action is completed)
- pagination (via the previous/next links).
  - Verify that the *current row number* and *total row count* are correct.
  - Check that pagination is reset after a table is refreshed (via column sort or pattern search) or the visualization changed.
- the commands available via the [Cog](#) icon particularly in the Page URLs report and in *Referrers > Websites*.
- the *Number of rows to display* dropdown.
- the inline search box. To test:
  - search for a string that is present in the label column of the table, navigate in the table and then cancel the search.
  - search for a string that is not present in the table (should display something like **There is no data for this report.**).
- the data exporting feature. Check that data can be exported as CSV, XML, JSON and TSV.*Note: You do not have to test for correctness since there are unit and integration tests that will test for that.*
- row evolution and multi row evolution.

TODO: what does 'navigate in the table' mean?

# Subtable Visualization

Check that:

- Clicking on a row displays a new datatable for the proper reports (eg, the Keywords table and the Search engine table) and verify all previous tests pass for this subtable (the same testing procedure as for the root table).

- There is an external url link on some subtable rows for reports such as the Search engines reports and the Keywords report.
- Row Evolution + Multi Row Evolution works in subtables.

## Other Report Visualizations

Check that switching to the following visualizations works:

- bar graph (should load a bar graph)
- pie chart (should load a pie graph)
- tag cloud (displays values with relative font size)
- table with all columns
- (optional) table with goal metrics

## Reports under *Actions > Pages*

Run the following tests for actions reports:

- Run the same tests that were run on the standard reports (exclude the test for alternate row styles for even/odd rows).
- Expand and minimize rows through several nesting levels (check that the left margin gets bigger each time).
- Check that the figures are summed correctly in the **include low pop** mode.

## Sparklines

Check that:

- On the *Visitors > Overview*, *Referrers > Overview* and *Goals > Overview* pages, check that clicking on a sparkline updates the graph above with the correct data.

## Goals plugin

Runn the following tests:

- Try to create/edit/delete Goals.
- In the main *Goals > Goals Overview* page, check that reports load properly when menu items on the bottom left (under **Conversions overview by type of visit**) are clicked.

# Dashboard

Run the following tests:

- Move widgets around (Flash and non-Flash content) TODO: flash still appropriate?
- Delete a widget.
- Add a new widget.
- Check that you can quit modal dialogs by pressing 'esc' key.
- Make sure that already added widgets cannot be added again.

# Calendar

Check that:

- The currently selected period is highlighted in the calendar.
- Changing the date and period works.
- Selecting a date range works.
  - Check that all widgets in the dashboards load w/ a date range as well.

# Menu

Check that:

- Clicking on a main menu item should select the first submenu item.
- The current menu category should be colored even when the mouse isn't over it.
- There are no broken links in the menu/submenus.

# Refresh and Back button

Run the following tests:

- Select a submenu, press refresh and check that the page refreshes as expected for the same submenu, the same period and the same website (ie, nothing changes).
- Change the website, check that the selected menu item page does not change.
- Select two different submenus, press Back button and check the state is restored as expected.

# Language Selector

Check that:

- Clicking on the language selector shows the list of languages.
- Clicking on a language from the list reloads Piwik and loads the clicked language.
- A click outside the language selector hides the open language selector.

If this works, it should also work for the Website Selector (which uses same code).

## Feedback

Check that:

- Clicking on the feedback link opens a box with grey background and links to forum and FA.
- Clicking on *Contact the Piwik team* should display a form with a dropdown and two input fields to send feedback.

Run the following tests:

- Send an invalid message (e.g., too short) and navigate back (using the left arrow in the feedback window)
- Test successfully sending a message.
- Close window; re-click on feedback link should open box with grey background and links to forum and FAQ; there should be no trace of the previous error/success message

## Widgetize

- Test one report visualization in Widgetize iframe mode.
- Test that the dashboard loads well in iframe mode.

## Custom Segment Editor

The custom segment editor is a critical piece of the Piwik User Interface. Login as the Super User and:

- Create a segment named "Hello ""worldend".
- Add a few AND/OR segments, drag and drop metrics from the left.
- Verify that the Auto Suggest fields are working on the INPUT values (should display a drop down of suggested values).
- Click on Save & Apply segment.
- Check that the **Loading data...** message is displayed, and a notice about custom segments taking a few minutes to process.
- Check that on page reload, the data for the segment is displayed. We recommend to test using *Visitors > Visitor Log* as it makes it easy to visualize the visits.

At this stage you have created a custom segment. Now you can test:

- *Visitors > Real time map*, check it displays only visitors from the segment.
- Click on *Email & SMS reports*. Add a new Email report. When creating the report, select the newly created segment.
  - Check that the Email report generated with the Custom Segment, contains the segment name at the top.
- Change the date in the calendar. Check that the selected segment does not change.

And maybe run some more advanced tests:

- Rename the segment name to a new name, and change one segment value to `@#$%^,&*('';<test>Test`. After clicking Save & Apply, click [edit], the segment name and value should display correctly.
- Check in a few reports that data is displayed for the Custom Segment
  - in the table report visualization, open the row evolution popover and check that the data loaded is for the segmented visitor set.
  - The [Transition report](#) should load for segmented data as well.

## Embed / Widgetize

In the top menu click on *Widgetize*.

- Test that Dashboard works when widgetized here.
- Test that other reports work when widgetized here.

## Testing the installer

To view the installation screens, rename the file

`config/config.ini.php`

to

`config/_config.ini.php`

Refresh Piwik: you will be prompted to install Piwik. Install in a new database or use a different prefix to go through the install steps.

## Testing the Updater

To test the updater process, first update the Piwik version number in the

`core/Version.php`

file, for example to 2.0-b100 (if you are using less than 2.0). Next, you need to create a new file in

```
core/Updates/2.0-b100.php  
, copying an existing Update file for inspiration (rename the class to  
Piwik_Updates_2_0_b100  
).
```

Refresh Piwik and you should see the Update screen. If you click "Update" and need to test the Updater again, you can update the version Piwik thinks it is using with the SQL query (source: [FAQ](#)):

```
UPDATE `piwik_option` SET option_value = "1.X" WHERE option_name = "version_core";
```

## Done!

If all these tests pass, the UI is working properly.

# Visualizing Report Data

---

## About this guide

### Read this guide if

- you'd like to know **how to display your reports the way existing reports are displayed**
- you'd like to know **the different ways your reports can be displayed**
- you'd like to know **how to get your report to display in the dashboard**
- you'd like to know **how to create new visualizations**

### Guide assumptions

This guide assumes that you:

- can code in PHP,
- have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide),
- and understand the purpose of [Piwik controllers](#) and [Piwik APIs](#).

# Displaying Analytics Reports

In Piwik, an analytics report is just a set of two-dimensional data (stored internally in [DataTable](#) instances). They are returned by API methods. Controller methods can output visualizations of these reports by using the [ViewDataTable](#) class.

Report visualizations can be in any format. The **sparkline** visualization for example outputs an image. Most visualizations, however, will output an HTML display that allows users to switch between different visualizations. You've very likely seen this display before:

TODO: image of datatable view

## About visualizations

A report visualization is a class that extends from [ViewDataTable](#). Every visualization has a unique ID that is specified by a class constant named **ID**. They are referred to by this ID, not their class name, because they must be able to be specified in the **viewDataTable** query parameter. Using a fully qualified class name would be too verbose.

The **viewDataTable** query parameter is set by [ViewDataTable](#)'s associated JavaScript code. It represents the user's choice of view (made by clicking on a visualization's footer icon) and thus will override any visualization type specified in code.

Users can switch visualizations by clicking on one of the icons displayed below the visualization itself:

TODO: image

These icons are called **footer icons**. Not all visualizations have to be available in this manner. For example, the [Visitor Log](#) uses its own visualization, but since it can only use data from one report ([Live.getLastVisitsDetails](#)) it is not available in the footer.

To learn more about the controls that surround a visualization in most report displays, [read the user docs for report visualizations](#).

## Using ViewDataTable

There are two ways to output a visualization for a report. The most succinct way is to call the [Controller::renderReport](#) method:

```
// controller method for the MyPlugin.myReport report
public function myReport()
```

```
{  
    return $this->renderReport(__FUNCTION__);  
}
```

[renderReport](#) will create a new [ViewDataTable](#) instance and render it. The report can be configured via the [ViewDataTable.configure](#) event.

The other way to output the display is to manually create and configure a [ViewDataTable](#) instance:

```
// controller method for the MyPlugin.myReport report  
  
public function myReport()  
{  
    $view = \Piwik\ViewDataTable\Factory::build(  
        $defaultType = 'table',  
        $apiAction = 'MyPlugin.myReport',  
        $controllerMethod = 'MyPlugin.myReport',  
    );  
    $view->config->show_limit_control = false;  
    $view->config->show_search = false;  
    $view->config->show_goals = true;  
  
    // ... do some more configuration ...  
  
    return $view->render();  
}
```

The visualization type is specified in the first argument to [\Piwik\ViewDataTable\Factory::build](#).

When [ViewDataTable::render](#) is called, the [ViewDataTable](#) instance will load a report through the reporting API (using the [PiwikAPI\Request](#) class) and output an HTML display using Twig templates.

*Note: The display generated by ViewDataTable will reload the report using AJAX, which means there must be a way to get the display for just the report in question. Therefore, you must create a dedicated controller method for each of your reports. You can use*

a `ViewDataTable` instance outside of a controller method, but it must still reference such a controller method.

### Configuring `ViewDataTable` instances

`ViewDataTable` instances are configured by setting properties of a `ViewDataTable`'s `Config` or `RequestConfig` object.

Properties in the `Config` object affect the way the report is displayed. Properties in the `RequestConfig` object affects how the data is obtained and, in part, what is displayed.

## Displaying reports on a page

Once there exists a controller method for a report, displaying it on a page in Piwik is straightforward. Assuming you've [exposed a controller method as a menu item](#), you can then [reuse your report's controller method](#) to include the report display in the menu item page:

```
// controller method exposed as a menu item

public function index()
{
    $view = new View("@MyPlugin/index.twig");
    $view->myReport = $this->myReport();
    echo $view->render();
}

// report method

public function myReport()
{
    return $this->renderReport(__FUNCTION__);
}
```

The `index.twig` template will look like this:

```
<h1>My Report</h1>

{{ myReport }}
```

# Displaying reports in the Dashboard

After a controller method is created for a report, the report can be made available to the dashboard by using the [WidgetsList.addWidgets](#) event:

```
// event handler for the WidgetsList.addWidgets event in the My
Plugin/MyPlugin.php file

public function addWidgets()
{
    WidgetsList::add('My Category Name', 'My Report Title', 'My
Plugin', 'myReport');
}
```

Piwik users will then be able to see and select **myReport** in the widget selector.

*Note: Any controller method can be embedded in the dashboard, not just reports. So if you have a popup that you'd like to make available as a dashboard widget, you can use [WidgetsList.addWidgets](#) to do so. This is exactly how we made the [Visitor Profile](#) available in the dashboard.*

## Core Visualizations

The following are a set of visualizations, called **Core Visualizations**, that come pre-packaged with Piwik (visualizations are listed by their **viewDataTable** ID):

TODO: IDs should link to images?

- **table**: This is the main visualization used to view report data. It is essentially a table of rows and columns.
- **tableAllColumns**: The same as **table** except a couple extra processed metrics are calculated and displayed using the report data.
- **tableGoals**: The same as **table** except goal metrics are processed and displayed for the report and each goal of the website the report is for.
- **graphVerticalBar**: Displays report data as a vertical bar graph. *Uses jqPlot.*
- **graphPie**: Displays report data as a pie graph. *Uses jqPlot.*
- **graphEvolution**: Displays a set of reports over time as a line graph. Metrics are displayed as different series.
- **cloud**: Displays report labels in a tag cloud using the values of a specified metric as significance.
- **sparkline**: Displays the values of one metric over time as a line graph. Outputs an image as opposed to HTML.

You can use these visualizations when creating [ViewDataTable](#) instances without having to install third-party plugins.

# Creating new Visualizations

Plugins can provide their own visualizations, either for use just within the plugin or as a new visualization that can be applied to any report. The [Treemap Visualization](#) plugin is an example of this:

TODO: treemap image

This section will explain how [ViewDataTable](#) works and everything you can do when extending it.

## Extending ViewDataTable

The first step in creating a new visualization is to create a class that descends from [ViewDataTable](#). If you are creating a visualization that will not output HTML, then extend directly from [ViewDataTable](#). If your visualization will be in HTML, extend from [Visualization](#).

[Visualization](#) is a direct descendant of [ViewDataTable](#) that provides common behavior you see in all core visualizations. That is, it shows the footer icons, the limit selector, the report documentation, the search box and more.

### Setting ViewDataTable metadata

In your new class that derives from [ViewDataTable](#) or [Visualization](#) the first thing you'll want to do is set the necessary visualization metadata. All visualization metadata are stored as class constants. You can set the following constants:

- **ID:** (*required*) The unique ID for this visualization. This is what gets supplied in the `viewDataTable` query parameter.
- **TEMPLATE\_FILE:** (*required*) A reference to a Twig template file, eg,

```
"@MyPlugin/_myDataTableViz.twig"
```

- **FOOTER\_ICON:** A path from Piwik's root directory to an icon to display in the footer of the visualization.
- **FOOTER\_ICON\_TITLE:** The tooltip to use when the user hovers over the footer icon. Can be a [translation token](#).

An example:

```
class MyVisualization extends Visualization
{
    const ID = 'myvisualization';
    const TEMPLATE_FILE = '@MyPlugin/_myVisualization.twig';
    const FOOTER_ICON = 'plugins/MyPlugin/images/myvisualization.png';
    const FOOTER_ICON_TITLE = 'My Visualization';
}
```

### Adding new display properties

Visualizations can define their own display properties (in addition to what is available in [Config](#)) by creating their own [Config](#) class. This new class must derive from [Config](#) and the visualization must provide an override for the [ViewDataTable::getDefaultConfig](#) method that creates an instance of this new class. For example:

```
class MyVisualizationConfig extends Config
{
    public $show_magic_widget = true;

    public $show_disclaimer = false;
}

class MyVisualization extends Visualization
{
    // ...

    public static function getDefaultConfig()
    {
        return new MyVisualizationConfig();
    }

    // ...
}
```

```

}

// method in MyPlugin's controller

public function myReport($fetch = false)
{
    $view = \Piwik\ViewDataTable\Factory::build(
        $defaultType = MyVisualization::ID,
        $apiAction = 'MyPlugin.myReport',
        $controllerMethod = 'MyPlugin.myReport',
    );

    // in a controller method somewhere
    $view = Factory::
        $view->config->show_limit_control = false;

    // set a new property
    $view->config->show_magic_widget = false;

    return $view->render();
}

```

Visualizations can also create their own [RequestConfig](#) class for properties that affect the request.

### Changing what data is loaded

Visualizations can alter exactly what data is loaded by the base [ViewDataTable](#) class if desired. The evolution graph visualization does this to get reports for the last N periods so it can display data over time. The [treemap visualization](#) does this to get the data last period so the percent change of each row can be displayed.

To change exactly what data is loaded, visualizations should alter the request by setting the[request parameters to modify](#) request config property within the[Visualization::beforeLoadDataTable](#) method. For example:

```
class MyVisualization extends Visualization
```

```

{

    // ...

    public function beforeLoadDataTable()
    {
        $date = Common::getRequestVar('date');

        $this->config->request_parameters_to_modify['date'] = Date::factory($date)->subDay(1)->toString();
    }

    // ...
}

```

*Note: If you change what data is loaded, you may also need to override the [Visualization::isThereDataToDisplay](#) method. Otherwise the [no data message](#) may not appear if there is no data in the report you are displaying.* TODO: no data message should link to image?Manipulating report data before displaying

Visualizations can modify report data after it is loaded by overriding one of the following methods:

- [Visualization::beforeGenericFiltersAreAppliedToLoadedDataTable](#): Called by [Visualization](#) after report data is loaded and after priority filters in the [filters](#) display property are executed. Called before [generic filters](#) are executed.

*Override this method if you need to use the report's full data (ie, before it has been limited/truncated/sorted/etc.).*

- [Visualization::afterGenericFiltersAreAppliedToLoadedDataTable](#): Called by [Visualization](#) after report data is loaded, after priority filters are executed and after [generic filters](#) are executed.

*Override this method if you need to use the report data after it has been sorted/filtered/limited/etc., but before presentation filters have been applied.*

- [Visualization::afterAllFiltersAreApplied](#): Called by [Visualization](#) after all filters (priority, generic and queued) are executed.

*Override this method if you need to use the report data after it is ready to be displayed.*

*Note: Report data can also be manipulated by setting the [filters](#) display property.*

### Making your visualization available to Piwik users

If you want to allow Piwik users to use your visualization on any report (by clicking the appropriate footer icon), add it to the global list of available visualizations in the [ViewDataTable.addViewDataTable](#) event:

```
// event handler for ViewDataTable.addViewDataTable in MyPlugin
plugin

public function addViewDataTable(&$visualizations)
{
    $visualizations[] = 'Piwik\\Plugins\\MyPlugin\\Visualizations\\MyVisualization';
}
```

Visualizations that are exposed this way must have the **FOOTER\_ICON** and **FOOTER\_ICON\_TITLE** [ViewDataTable metadata](#) set.

## Extending ViewDataTable's JavaScript

Extending [ViewDataTable](#) is only half the battle. Unless your visualization is very simple or builds on another visualization (such as the **tableAllColumns** and **tableGoals** visualizations) you will probably need to extend ViewDataTable's client-side logic.

All of [ViewDataTable](#)'s client side logic is encapsulated within the **DataTable** JavaScript class (located in the [dataTable.js](#) file). If you are extending [Visualization](#) directly, you should extend **DataTable** to add your own client-side logic. If you're extending another visualization, you will have to extend that visualization's JavaScript class. For example:

```
(function ($, require) {

    var exports = require('piwik/UI'),
        DataTable = exports.DataTable,
        dataTablePrototype = DataTable.prototype;

    /**
     * ...
     */
})()
```

```

* Class that handles UI behavior for the MyVisualization visualization.

*/
exports.MyVisualization = function (element) {
    DataTable.call(this, element);
};

$.extend(exports.MyVisualization.prototype, dataTablePrototype, {
    // ...
});

}(jQuery, require));

```

When extending **DataTable** the only method you are required to override is the **init** method. Here, you should place your visualization's initialization code:

```

$.extend(exports.MyVisualization.prototype, dataTablePrototype,
{

    init: function () {
        dataTablePrototype.init.call(this);

        this._bindEventCallbacks(this.$element);
        this._addSeriesPicker(this.$element);

        // ... etc. ...
    }
})

```

After you extend the class, you must notify [Visualization](#) of the new class by setting the [datatable\\_js\\_type](#) display property in your visualization. For example:

```

class MyVisualization extends Visualization
{

```

```

// ...

public function beforeRender()
{
    $this->config->datatable_js_type = 'MyVisualization';

}

// ...
}

```

## Adding new UI controls

Some visualizations will require UI controls that are not in the base report display. The graph visualizations included with Piwik, for example, add a widget called the **series picker** to the UI:

TODO: image of all three graphs w/ metric picker

New controls can be added one of two ways. They can be added entirely through JavaScript (as is done with the **series picker**) or the HTML for a control can be rendered with a visualization and the visualization's JavaScript class can attach some logic to the HTML.

**To add controls entirely in JavaScript**, create HTML elements dynamically within your visualization's JavaScript class' **init** method. See [the series picker source code](#) for an example.

**To add controls without having to create HTML elements in JavaScript**, add HTML to your visualization's twig template then bind logic to the HTML in your visualization's JavaScript. See [the treemap plugin](#) for an example (the zoom-out button is an extra control).

## Styling your visualization

The root

```
<div>
```

of every report visualization will have the **dataTable** CSS class. They will also have a CSS class based on the visualization used. It will look like **dataTableVizMyVisualization** where **MyVisualization** will be the unnamespaced class name of the visualization (for example **dataTableVizHtmlTable**).

You can select all report visualizations with the

```
.dataTable
```

CSS selector. You can select report visualizations of a specific type using the visualization specific CSS class, for example,

```
.dataTableVizHtmlTable
```

or

```
.dataTableVizHtmlTable > .dataTableWrapper
```

You can also use the [`datatable\_css\_class`](#) property to add more CSS classes if you need to. This property can be useful if you need to customize the way visualizations appear for specific reports.

## Making your visualization theme-able

To make sure your visualization can be themed, make sure any color value you use in your JavaScript is obtained from CSS. Read about how we accomplish this [here](#).

## Learn more

- To learn **how reports are stored and created**, read our [All About Analytics](#) guide.
- To see a **full example of creating a new visualization**, see the source for the [Treemap Visualization](#) plugin.
- To learn more about **Piwik Controllers and outputting HTML**, read our [MVC in Piwik](#) guide.
- To learn more about **interacting with Piwik's client side JavaScript**, read our [Working with Piwik's UI](#) guide.

## Working with Piwik's UI

### About this guide

This guide describes how plugins should create JavaScript and describes all of the JavaScript classes provided by Piwik for use by plugins.

#### Read this guide if

- you'd like to know **how to write JavaScript for your Piwik plugin**
- you'd like to know **how to work with Piwik Core's JavaScript code**
- you'd like to know **how to add popovers to Piwik's UI**
- you'd like to know **what JavaScript libraries are used by Piwik**

## Guide assumptions

This guide assumes that you:

- can code in JavaScript,
- can use [jQuery](#),
- know what CSS and [LESS](#) is,
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

# JavaScript Libraries Piwik depends on

Piwik Core depends on the following JavaScript libraries:

- [jQuery](#)
- [jqPlot](#)

## New Library Policy

We do not like to include new dependencies in Piwik, so if possible do not include new third party libraries in your contribution or plugin.

This suggestion is important for contributions, but doubly important for plugins. Imagine what would happen if almost every plugin decides they need to use a library that Piwik Core does not use. Users might install a couple plugins in the marketplace and end up loading several different new libraries on page load, slowing down the UI.

**Include new JS libraries only if they are vital to your plugin.**

# JavaScript, CSS and LESS file inclusion

JavaScript, CSS and LESS assets only exist in plugins. Piwik's core code (everything in the

`core/`

subdirectory) does not contain or define any asset files.

Plugins tell Piwik about their asset files through the [AssetManager.getStylesheetFiles](#) and [AssetManager.getJavaScriptFiles](#) events. Each event passes an array by reference to event handlers. Event handlers should add paths to assets to the array:

```
// event handler for AssetManager.getStylesheetFiles
public function getStylesheetFiles(&$files)
```

```

{
    $files[] = "plugins/MyPlugin/stylesheets/myStylesheet.less";
    $files[] = "plugins/MyPlugin/stylesheets/myCssStylesheet.css";
}

// event handler for AssetManager.getJavaScriptFiles
public function getJavaScriptFiles(&$files)
{
    $files[] = "plugins/MyPlugin/javascripts/myJavaScript.js";
}

```

## Asset merging and compiling

In production environments, Piwik will concatenate all JavaScript into one file and minify it before serving it. LESS files will automatically be compiled into CSS and merged into one CSS file. Piwik does this in order to keep loading the UI fast. Learn more about asset merging in [this blog post](#).

JavaScript is merged only when enabling or disabling a plugin or theme. LESS compilation and merging is done whenever a LESS file changes (does not include LESS files that are included by others).

If the

[Debug] disable\_merged\_assets

INI config option is set to

1

, assets will not be merged. It can be useful to disable merged assets while developing a contribution or plugin since changes to JavaScript will then appear immediately.

TODO: make sure getting started talks about disabling asset merging

## JavaScript modularization

Piwik attempts to modularize all JavaScript code through the use of Piwik's

**require**

function. This function (inspired by [node.js' require function](#)) will create nested objects in the **window**

object based on a namespace string (for example,

```
'MyPlugin/Widgets/FancySchmancyThing'
```

).

Here's how it should be used:

```
(function (require, $) {  
  
    // get a class that we're going to extend  
    var classToExtend = require('AnotherPlugin/Widgets').TheirWidget;  
  
    // extend it  
    function MySpecialWidget() {  
        classToExtend.call(this);  
    }  
  
    $.extend(MySpecialWidget.prototype, classToExtend.prototype, {  
        // ...  
    });  
  
    // export the class so it is available to other JavaScript files  
    var exports = require('MyPlugin/NamespaceForThisFile');  
    exports.MySpecialWidget = MySpecialWidget;  
  
})(require, jQuery);
```

All new JavaScript should modularize their code with the

```
require
```

```
function. The
```

```
window
```

```
object should be accessed sparingly, if ever.
```

# Special HTML Elements

The following is a list of special elements that you should be aware of as you develop your plugin or contribution:

- **#root**: The root element of everything that is displayed and visible to the user.
- **#content**: The root element that contains everything displayed under the main reporting menu and the row of 'selector' controls (ie, the period selector, the segment selector, etc.).
- **.top\_controls**: The element that contains the 'selector' controls. Only one of these elements.
- **.widgetContent**: The root element of each widget. Events are posted to this specific element.

# Important JavaScript classes

Piwik Core defines many classes that should be reused by new plugins and contributions. These classes can be used to, among other things, change the page the UI shows, load a popover and get themed color values from CSS.

## Broadcast

The

`broadcast`

object is stored directly in the

`window`

object and should be used to parse the current URL, load a page in the area below the menu and load persistent popovers.

*Note: Though the object is stored in*

`window`

*and not a JS namespace, it can still be accessed using*

`require`

*by calling*

`require('broadcast')`

## Parsing the URL

`broadcast`

contains several methods for parsing the Piwik URL. Piwik's main UI stores secondary set of

query parameters in a URL's hash value. This is the URL that is loaded below the menu.

### broadcast

's URL parsing functions will look for query parameter values in the hash as well as the main query string, so it is important to use them instead of directly accessing

### window.location.href

### broadcast

provides the following functions:

- **isHashExists**: Returns the hash of the URL if it exists.

### false

if otherwise.

- **getHashFromUrl**: Returns the hash of the URL. Can be an empty string.
- **getSearchFromUrl**: Returns the query string.
- **extractKeyValuePairsFromQueryString**: Converts a query string to an object mapping query parameter names with query parameter values.
- **getValuesFromUrl**: Returns an object mapping query parameter names with query parameter values for the current URL.
- **getValueFromUrl**: Returns one query parameter value for the current URL by name.
- **getValueFromHash**: Returns one query parameter value in the hash of the current URL by name.
- **getHash**: Returns the hash of the URL.

To learn more about an individual function, see the method documentation in [theplugins/CoreHome/javascripts/broadcast.js](#) file. TODO: document broadcast.js functions better

## Loading new Piwik pages

To load a new page below the main Piwik menu, use the **propagateNewPage** function with a URL to the controller method whose output should be displayed:

```
(function (require) {  
  
    var broadcast = require('broadcast');  
  
    broadcast.propagateNewPage("index.php?module=MyPlugin&action=mySpecialPage", true);  
  
})(require);
```

## Loading Persistent Popovers

To load a popover that will be displayed even if the page is reloaded, you'll need to call two functions. Piwik makes a popover *persistent* by adding a **popover** query parameter. The parameter value will contain a popover ID and another string (separated by a

```
' : '
```

). Piwik will see this ID and execute a function that displays the popover.

The first method you need to call is named **addPopoverHandler**. It associates a function with the popover ID. The function will be passed the rest of the popover query parameter. For example:

```
(function (require) {  
  
    var broadcast = require('broadcast');  
  
    broadcast.addPopoverHandler('myPopoverType', function (arg)  
    {  
        Piwik_Popover.createPopupAndLoadUrl("?module=MyPlugin&action=getPopup&arg=" + arg, _pk_translate('MyPlugin_MyPopoverTitle'));  
    });  
  
})(require);
```

Then, when you want to launch a popover, call the **propagateNewPopoverParameter** method:

```
(function (require, $) {  
  
    var broadcast = require('broadcast');  
  
    $('#myLink').click(function (e) {  
        e.preventDefault();  
  
        broadcast.propagateNewPopoverParameter('myPopoverType',  
        'myarg');  
  
        return false; getPageUrls  
    });  
})(require);
```

```
});  
  
})(require, jQuery);
```

## ajaxHelper

The

### ajaxHelper

class should be used whenever you need to create an AJAX request. **Plugins should not use `$.ajax` directly.**

### ajaxHelper

does some extra things that make it harder to write insecure code. It also keeps track of the current ongoing AJAX requests which is vital to the [UI tests](#).

To use the

### ajaxHelper

, create an instance, configure it, and then call the `send` method. To learn more, read the documentation in the source code (located in `plugins/Zeitgeist/javascripts/ajaxHelper.js`).

An example:

```
(function (require, $) {  
  
    var ajaxHelper = require('ajaxHelper');  
  
    var ajax = new ajaxHelper();  
    ajax.setUrl("index.php?module=Actions&action=getPageUrls&id  
Site=1&date=today&period=day");  
    ajax.setCallback(function (response) {  
        $('#myReportContainer').html(response);  
    });  
    ajax.setFormat('html'); // the expected response format  
    ajax.setLoadingElement('#myReportContainerLoading');  
    ajax.send();  
  
})(require, jQuery);
```

TODO: change name of ajaxHelper class to AjaxHelper?

## UIControl

### UIControl

is meant to be the base type of all JavaScript widget classes. These classes manage and attach behavior to an element. Examples of classes that extend from UIControl include [DataTable](#)(the base of all [report visualization](#) JavaScript classes) and **VisitorProfileControl** (used to manage the [visitor profile](#)).

### UIControl

allows descendants to clean up resources, provides a mechanism for server side code to send information to the UI and provides a method of listening to dashboard widget resize events.

## Extending UIControl

The actual extending is straightforward:

```
(function (require, $) {  
  
    var UIControl = require('piwik/UI').UIControl;  
  
    var MyControl = function (element) {  
        UIControl.call(this, element);  
  
        // ... setup control ...  
    };  
  
    $.extend(MyControl.prototype, UIControl.prototype, {  
        // ...  
    });  
  
    var exports = require('MyPlugin');  
    exports.MyControl = MyControl;  
})(require, $);
```

### UIControl

's constructor takes one argument, the HTML element that is the root element of the widget.

## Creating controls that extend UIControl

Control instances should be created through the **initElements** static method:

```
MyControl.initMyControlElements = function () {  
    UIControl.initElements(this, '.my-control');  
};
```

This will find all elements with the **my-control** class, and if they do not already have a **MyControl** instance associated with them, it will create an instance with that element.

**MyControl.initMyControlElements** should be called when your control's HTML is added to the DOM. This is often done in Piwik by including a **<script>** element in HTML returned by AJAX, for example:

```
<div class="my-control">  
</div>  
<script type="text/javascript">require('MyPlugin').MyControl.in  
itMyControlElements();</script>
```

## Cleaning up after your control

When the selected page changes or when a popover is closed, Piwik will call the **UIControl.cleanupUnusedControls** static method. This method will automatically collect all control instances that are attached to elements that are not part of the DOM and call the controls' **\_destroy** method.

When creating your own control, if you need to do some extra cleanup, you can override this method:

```
$.extend(MyControl.prototype, UIControl.prototype, {  
  
    _destroy: function () {  
        UIControl.prototype._destroy.call(this);  
  
        this.myThirdPartyLibWidget.destroy();  
    }  
  
});
```

## Sending information from PHP to UIControl

If you need to pass information from PHP code to

UIControl

instance, you can set the **data-propsHTML** attribute of the root element of your control to a JSON string. This data will automatically be loaded and stored in the **props** attribute of a

UIControl

instance.

So if you create HTML like the following:

```
<div class="my-control" data-props="{"title": "My Control"}>
</div>
<script type="text/javascript">require('MyPlugin').MyControl.initMyControlElements();</script>
```

then

`this.props.title`

will be set to

'My Control'

:

```
var MyControl = function (element) {
    UIControl.call(this, element);

    alert(this.props.title); // will say 'My Control'
};
```

## Listening to dashboard widget resize

To redraw or resize elements in your control when a widget is resized, call the **onWidgetResize** method when setting up your control:

```
var MyControl = function (element) {
    UIControl.call(this, element);

    var self = this;
    this.onWidgetResize(function () {
        self._resizeControl(); // private method not shown
    });
};
```

```
});  
};
```

## Piwik\_Popover

The **Piwik\_Popover** object is stored directly in the

```
window
```

object and contains popover creation and management functions. Popovers created directly through this object are not persistent. To create persistent popovers, use the [broadcast object](#).

To learn more about the object, see the documentation in the source code (located in `plugins/CoreHome/javascripts/popover.js`).

### Creating popovers

To create a popover, use the **createPopupAndLoadUrl** method:

```
(function (require) {  
  
    var Piwik_Popover = require('Piwik_Popover');  
  
    Piwik_Popover.createPopupAndLoadUrl("?module=MyPlugin&action=getMyPopover", "The Popover Title", 'my-custom-dialog-css-class');  
  
})(require);
```

Creating a popover will close any popover that is currently displayed. Only one popover can be displayed at a time.

### Closing popovers

To close the currently displayed popover, call the **close** method:

```
(function (require) {  
  
    var Piwik_Popover = require('Piwik_Popover');  
    Piwik_Popover.close();  
  
})(require);
```

## ColorManager

If your control uses color values to, for example, draw in canvas elements, and you want to make those colors [themeable](#), you must use the **ColorManager** singleton.

JavaScript colors are stored in CSS like this:

```
.my-color-namespace[data-name=my-color-name] {  
    color: red;  
}
```

In your JavaScript, you can use **ColorManager** to access these colors:

```
(function (require) {  
  
    var ColorManager = require('piwik').ColorManager;  
  
    // get one color  
    var myColorToUse = ColorManager.getColor('my-color-namespace', 'my-color-name');  
  
    // get multiple colors all at once  
    var myColorsToUse = ColorManager.getColor('my-color-namespace', ['my-first-color', 'my-second-color']);  
  
})(require);
```

To learn more about the singleton, read the source code documentation (located in [plugins/CoreHome/javascripts/color\\_manager.js](#)).

*Learn more about theming in our [Theming](#) guide.*

## DataTable

The **DataTable** class is the base of all JavaScript classes that manage [report visualizations](#). If you're creating your own report visualization, you may have to extend it.

To learn more about extending the class, see our [Visualizing Report Data](#) guide.

TODO: should I talk about everything the class does here? guide could get much longer.

# Global variables defined by Piwik

Piwik defines several global variables (held in

`window.piwik`

) regarding the current request. Here is what they are and how you should use them:

- `piwik.token_auth`

: The [token auth](#) of the current user. Should be used in AJAX requests, but should never appear in the URL.

- `piwik.piwik_url`

: The URL to this Piwik instance.

- `piwik.userLogin`

: The current user's login handle (if there is a current user).

- `piwik.idSite`

: The ID of the currently selected website.

- `piwik.siteName`

: The name of the currently selected website.

- `piwik.siteMainUrl`

: The URL of the currently selected website.

- `piwik.language`

: The currently selected language's code (for example,

`en`

`).`

- `piwik.config.action_url_category_delimiter`

: The value of the

`[General] action_url_category_delimiter`

INI config option.

# Coding conventions

When writing JavaScript for your contribution or plugin, be sure to follow the following coding conventions.

## Self-executing anonymous function wrapper

Every JavaScript file you create should surround its code in a self-executing anonymous function:

```
/**  
 * My JS file.  
 */  
(function (require, $) {  
  
    // ... your code goes here ...  
  
})(require, jQuery);
```

If you need to use global objects, they should be passed in to the anonymous function as is done above. Anything that should be made available to other files should be exposed via [require](#).

## Private methods

Prefix all private and protected methods in classes with an `_`:

```
MyClass.prototype = {  
  
    myPublicFunction: function () {  
        // ...  
    },  
  
    _myPrivateFunction: function () {  
        // ..  
    }  
}
```

```
};
```

## Learn more

- To learn **about creating new report visualizations** read our [Visualizing Report Data](#) guide.
- To learn **more about the asset merging system** read this [blog post](#) by the system's author.
- To learn **more about theming** read our [Theming](#) guide.

# Piwik Configuration

## About this guide

### Read this guide if

- you'd like to know **how your plugin can define its own configuration settings**
- you'd like to know **how the INI configuration system works**

### Guide assumptions

This guide assumes that you:

- can code in PHP,
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

# Piwik Configuration

Piwik uses methods to store configuration settings, the INI files in the **config** folder and **Options** which are persisted to the database. These methods are used by **Piwik Core** and should not be used by plugins. Plugins use a separate method of configuration [described below](#).

## The INI files

INI configuration options are stored in both the **config/config.ini.php** and **config/global.ini.php** files. The **global.ini.php** file

contains the default values for configuration options. If an option does not exist in **config.ini.php** the value in **global.ini.php** is used.

Plugins and other code can access and modify INI config options using the [Piwik\Config](#) class.

*To learn more about individual configuration options, read the documentation in [global.ini.php](#).*

## Options

Some Piwik configuration settings are stored as **Options**. **Options** are just key value pairs that are persisted in the database. To learn more about options, read the docs for the [Option](#) class.

TODO: this helps people w/ distributed setups correct? need to find out how

\_To learn about how options are persisted in the MySQL backend Piwik uses, read our [Persistence & the MySQL Backend](#) guide.

## Plugin Configuration

Plugins can define their own configuration options by creating a class named **Settings** that extends [Piwik\Plugin\Settings](#). The subclass should implement the [Settings::init](#) method adding settings that can be set by the user. For example,

```
class Settings extends \Piwik\Plugin\Settings
{
    /**
     * @var UserSetting */
    public $autoRefresh;

    /**
     * @var UserSetting */
    public $refreshInterval;

    protected function init()
    {
        $this->setIntroduction('Here you can specify the settings for this plugin.');
    }
}
```

```
$this->createAutoRefreshSetting();

$this->createRefreshIntervalSetting();

}

private function createAutoRefreshSetting()
{
    $this->autoRefresh      = new UserSetting('autoRefresh',
        'Auto refresh');

    $this->autoRefresh->type = static::TYPE_BOOL;

    $this->autoRefresh->uiControlType = static::CONTROL_CHECKBOX;

    $this->autoRefresh->description = 'If enabled, the value will be automatically refreshed depending on the specified interval';

    $this->autoRefresh->defaultValue = false;

    $this->addSetting($this->autoRefresh);
}

private function createRefreshIntervalSetting()
{
    $this->refreshInterval      = new UserSetting('refreshInterval',
        'Refresh Interval');

    $this->refreshInterval->type = static::TYPE_INT;

    $this->refreshInterval->uiControlType = static::CONTROL_TEXT;

    $this->refreshInterval->uiControlAttributes = array('size' => 3);

    $this->refreshInterval->description = 'Defines how often the value should be updated';

    $this->refreshInterval->inlineHelp = 'Enter a number which is >= 15';

    $this->refreshInterval->defaultValue = '30';
}
```

```

    $this->refreshInterval->validate = function ($value, $setting) {
        if ($value < 15) {
            throw new \Exception('Value is invalid');
        }
    };

    $this->addSetting($this->refreshInterval);
}

}

```

Plugins that do this will cause new sections to appear in the *Settings > Plugins* admin page:

TODO: image of result of above code

TODO: how do you access plugin settings?

See the [ExampleSettingsPlugin](#) to see what other things can be done.

# Distributing Your Plugin

---

## About this guide

**Read this guide if**

- you've created a plugin and want to share it with other Piwik users
- you'd like to know how you can take advantage of the Piwik marketplace

### Guide assumptions

This guide assumes that you:

- know how to use [git](#)
- and have already created a Piwik plugin (if not, start by reading our [Getting Started Extending Piwik](#) guide).

# The Marketplace

Piwik's [plugin marketplace](#) is the main way to download and install third-party plugins. Every instance of Piwik running version 2.0 or greater is able to directly download and install plugins from the marketplace. (TODO: will this be true by 2.0?) **Making your plugin available on the marketplace is the best way for you to get your plugin out into the hands of Piwik users.**

It's also a great way for you to:

- see how many people use your plugin
- allow people to donate money
- get exposure for your skills and work
- get feedback in the form of comments and ratings (*comming soon*)

## Adding your plugin to the marketplace

Getting your completed plugin on the marketplace takes a couple steps, all listed below.

### Make sure your plugin has a unique name

Every plugin on the marketplace has a unique name. If your plugin's name is already taken, you won't be able to put it on the marketplace.

Make sure the name you chose is currently available, and if it's not, pick another one.

TODO: problem w/ prefixing project names w/ user names so projects can have the same name?

### Getting your plugin ready to publish

Two files are required to be present in your plugin before you can publish: the **README.md** file and the [plugin.json](#) file. The **README.md** file should contain a description of your plugin and some documentation.

The **plugin.json** file must contain the following information:

- **name** - The plugin's name. It can only contain letters, numbers and the



and



characters. The name cannot contain the words

"Piwik"

,

"Core"

or

"Analytics"

- **version** - The plugin's version. It must be a valid semantic version number. If [node-semver](#) can't parse it, it won't be considered valid.
- **description** - A short description of your plugin (up to 150 characters). This will be displayed below the plugin's name in search results and below the top-level heading on your plugin's page. It can include any character.
- **keywords** - An array of words or short phrases that describes your plugins. The keywords are listed on the Marketplace, which helps users discover your plugin. Keywords can only contain letters, numbers, hyphens, and dots.
- **license** - The name of the license your plugin uses. The license must be compatible with the[GPLv3](#) or later. We recommend using [GPLv2](#) or later.
- **homepage** - The url to the plugin's homepage.
- **authors** - An array of objects, each describing someone who helped create this plugin. The objects must contain a **name** field and can optionally contain an email and homepage field, for example:

```
"authors": [ { "name": "Hacker", "email": "marketplace@piwik.org", "homepage": "http://piwik.org" } ]
```

You must define at least one author.

The following fields are not required for publishing a plugin, but you may want to add them:

- **donate** - An object containing information on how to donate to the plugin author (you!). The object can contain any of the following fields:
  - **paypal** - Your paypal email address.
  - **flattr** - The URL to your [Flattr](#) page.
  - **bitcoin** - Your Bitcoin address.

For example:

```
"donate": {  
    "paypal": "supporters@piwik.org",  
    "flattr": "https://flattr.com/thing/131552/Piwik-Web-Anal  
ytics-Open-Source",  
    "bitcoin": "http://piwik.org"  
}
```

## Put your plugin on Github

The marketplace uses [Github](#) webhooks to learn about your plugin and serve it to Piwik users that want it. This means you don't manually upload any files to the marketplace. Instead, you put your code into a Github repository and let the marketplace know about it.

Creating and initializing a [git](#) repo on Github is out of scope for this guide. If you need to learn how to create a Github repo, [read this article](#).

## Activating the Piwik Plugins webhook

Once you have a Github repo for your plugin, you need to let the marketplace know about it. This is done by activating the Piwik Plugins webhook.

To activate this webhook, follow these steps (TODO: show images):

1. Go to your plugin's Github repo in a browser.
2. Click on **Service Hooks**.
3. Scroll down until you see the **Piwik Plugins** option and then click on it.
4. Click the **Active** checkbox.
5. Click the **Update Settings** button.

The marketplace will now be notified every time you push a commit or a tag to your repository.

## Publish the first version of your plugin

Now that the marketplace will know of any changes to your plugin, you can publish the first version. It takes two git commands to publish a version of your plugin:

```
$ git tag 0.1.0
```

```
$ git push origin --tags
```

Everytime you push a new tag to your Github repository, a new version of your plugin will become available in the marketplace. The name of the tag doesn't matter, the marketplace will always use the version in your **plugin.json** file.

**Assuming all goes well, your plugin should now be visible on the marketplace!**

### Troubleshooting

If your plugin is not on the marketplace, then there was an error validating your **plugin.json** file and you will receive an email describing what the problem was.

Here are some common errors:

- The **plugin.json** manifest file was not found.
- Some of **plugin.json**'s [required fields](#) are not set.
- The version in **plugin.json** has already been published for this plugin.
- The **plugin.json** file does not contain valid JSON.
- The **README.md** file is missing.
- A plugin with your plugin's name already exists on the marketplace.
- There is a PHP syntax error in your plugin.

If you still encounter trouble while publishing your plugin, please join the IRC channel **#piwik** on[freenode](#). If you can't find anyone in the IRC channel, please ask for help [on the forums](#).

### Rules for Plugins

There are some restrictions regarding what can be published on the marketplace. Chances are your plugin is fine, but if we find a plugin that violates one of the following rules, it will be immediately removed:

- Your plugin must not do anything illegal, or be morally offensive.
- Your plugin's license must be compatible with the [GNU General Public License v3](#) or any later version. We strongly recommend using the same license as Piwik (*GPLv3 or later*).
  - **Note:** If you don't specify a license anywhere in your plugin, it is assumed your plugin uses GPLv3 or later.
- Your plugin must not contain obfuscated code. We believe that obfuscated code violates the spirit, if not the letter, of the GPL license under which we operate.
- No [phoning home](#) without user's informed consent. For the purposes of a Piwik plugin, **phoning home** includes:
  - No unauthorized collection of user data. For example, sending the admin's email address back to your own servers without the user's permission is not allowed; but asking the user for an email address and collecting it if they

- choose to submit it is fine. All actions taken in this respect MUST be of the user's doing, not automatically done by the plugin.
- All images and scripts shown should be part of the plugin. These should be loaded locally. If the plugin requires data that is loaded from an external website (such as [blocklists](#)) this should be made clear in the plugin's admin screens or its description. The user must be informed of what information is being sent where.
  - The plugin page, contents of the **readme.md** file and all translation files may not have "sponsored" links on it. This applies to all content that will be displayed on [plugins.piwik.org](#) and [themes.piwik.org](#)
  - The plugin cannot violate our [trademarks](#). Do not use **piwik** in your domain name, instead come up with your own original branding! People remember names.

## Using the marketplace

Now that you've gotten your plugin on the marketplace, it's time to learn how to use it. This section will explain how you can get the most out of the marketplace.

### Your plugin's page

Every plugin gets its own page on the marketplace. On the left is the name and a short description of your plugin followed by a set of tabs:

TODO: image

The content of the tabs is determined by the headings in your **README.md** file. See this [README.md](#) for an example.

On the right side is the download link, information about your plugin and some relevant links:

TODO: image

## Our plans for the marketplace

Like Piwik, the marketplace is a constant work in progress! We will be improving and changing it often to make it easier for you to find users for your plugin.

Here are the things we are currently working on:

- Allowing users to leave feedback through comments on plugin pages.
- Allowing users to rate existing plugins.
- TODO: what else?

# Piwik's Extensibility Points

---

## About this guide

**Read this guide if**

- you'd like to know **more about Piwik's event system**,
- you'd like to know **of every way you can extend Piwik with a plugin**.

### Guide assumptions

This guide assumes that you:

- can code in PHP,
- and have a general understanding of extending Piwik (if not, read our [Getting Started](#) guide).

## Piwik Extensibility

Plugins can extend Piwik using the following methods:

- using the event emitting and handling system
- implementing special classes that are recognized by Piwik
- extending certain abstract base classes

This guide provides an overview describing how plugins can use these methods and how Piwik provides these methods.

## The Event Dispatching System

At certain points during Piwik's execution, Piwik will signal to the EventDispatcher that the callbacks for a certain event should be invoked. This is called **posting an event**.

Events are posted with an array of variables that are passed on to each callback as arguments. These variables can be passed by value or by reference.

Plugins can associate callbacks (sometimes called **event handlers**) with events, and are therefore able to insert custom logic into different parts of Piwik. Plugins can also use events to notify Piwik of new implementations of [extendable classes](#).

You can view the list of all events Piwik will post [here](#).

# Handling Events

Events are **handled** or **observed** by associating a callback with an event. There are two ways to do this:

1. Plugins can map a callback by name in the [Plugin::getListHooksRegistered](#) method of their plugin descriptor class, for example:

```
``` class MyPlugin extends \Piwik\Plugin { public function getListHooksRegistered()
{ return array( 'API.getSegmentsMetadata' => 'getSegmentsMetadata',
'SomeClass.OtherEvent' => function ($arg1, $arg2) { // ... } ); }

public function getSegmentsMetadata(&$result) { $result[] = array( // ... ); } }````
```

2. Or plugins can call the [Piwik::addAction](#) method, for example:

```
3. Piwik::addAction('API.getSegmentsMetadata', function (&$result) {
4.   // ...
});
```

## Callback Execution Order

Callbacks can be made to run before or after others. This is done when associating callbacks with events.

To make a callback execute before all others, associate the event with an array like the following:

```
public function getListHooksRegistered()
{
    return array(
        'API.getSegmentsMetadata' => array(
            'before' => true,
            'function' => 'getSegmentsMetadata'
        )
    );
}
```

To make a callback execute after other callbacks, associate the event with an array like the following:

```
public function getListHooksRegistered()
{
    return array(
        'API.getSegmentsMetadata' => array(
            'after' => true,
            'function' => 'getSegmentsMetadata'
        )
    );
}
```

## Posting Events

Plugins can post events themselves if they want to be able to be extended by other plugins. The [ScheduledReports](#) plugin for example does this to [allow plugins to define new transport mediums and report formats](#).

To post an event, call the [Piwik::postEvent](#) function using the name of your event:

```
Piwik::postEvent('MyPluginOrClass.MyEvent', array($myFirstArg,
&$myRefArg));
```

### Event Naming Conventions

Event names should follow this format:

```
"$scopeName.$shortEventDescription"
```

where

**\$scopeName**

is either the name of the plugin that is posting the event or the name of the class and where

**\$shortEventDescription**

is a short description of the event's purpose, for example, `getAvailableViewDataTables`.

## Posting Events in Templates

You can post events within Twig templates by using the **postEvent** function:

```
{{ postEvent('MyPlugin.MyEventInATemplate') }}
```

The **postEvent** function will pass a string by reference to all event handlers and then insert the string into the template. Event handlers can modify the string, inserting HTML into templates in other plugins:

```
class MyOtherPlugin extends \Piwik\Plugin
```

```

{

    public function getListHooksRegistered()
    {
        return array(
            'MyPlugin.MyEventInATemplate' => 'handleMyEventInATemplate'
        );
    }

    public function handleMyEventInATemplate(&$outString)
    {
        $outString .= '<h1>This text was injected!</h1>';
    }
}

```

## Plugin Classes

Plugins can define certain special classes in order to extend Piwik. These classes must be placed in the root namespace of the plugin. Piwik will automatically detect, instantiate and use them.

## API and Controller

Plugins can define an **API** class (that extends [Piwik\Plugin\API](#)) to add more methods to the[Reporting API](#). They can also define a **Controller** class to handle HTTP requests that are sent by Piwik's UI.

*Learn more about these classes in our [MVC in Piwik guide](#).*

TODO: is the base class(es) for Controller classes specified somewhere?

## Archiver

Plugins can define an **Archiver** class (that extends [Piwik\Plugin\Archiver](#)) to hook into Piwik's[Archiving Process](#). These classes are used to calculate and cache analytics data. They are, thus, very important types.

*Learn more about these classes in our [All About Analytics](#) guide.*

# Settings

Plugins can define a **Settings** class (that extends [Piwik\Plugin\Settings](#)) to define their own configuration settings. These classes add sections to the *Settings > Plugins* admin page.

*Learn more about these classes in our [Piwik Configuration](#) guide.*

# Extendable Classes

Some abstract classes defined by Piwik can be extended to provide different behavior. These classes usually provide some way for users to choose between different implementations.

## LocationProvider

The [LocationProvider](#) class (defined in the UserCountry plugin) can be extended to provide new means of geolocation. Currently, the default implementation will guess a visitor's country based on the language their browser is using. UserCountry provides three other implementations that use GeoIP databases in different ways.

## ViewDataTable & Visualization

The [ViewDataTable](#) class can be extended by plugins that want to provide new [report visualizations](#). New subclasses must be registered with Piwik through the [ViewDataTable.addViewDataTable](#) event.

*Learn more about creating new report visualizations in our [Visualizing Report Data](#) guide.*

## Learn More

- To learn **about every event that Piwik posts** [read the event docs](#).
- To learn **more about the Twig filters and functions Piwik defines** read the documentation for the [View](#) class.
- To learn **about API and Controller classes** read our [MVC in Piwik](#) guide.
- To learn **about Archiver classes** read our [All About Analytics](#) guide.
- To learn **about plugin Settings** read our [Piwik Configuration](#) guide.
- To learn **about creating new LocationProviders** read our ???
- To learn **about creating new report visualizations** read our [Visualizing Report Data](#) guide.\_

# Persistence & the MySQL Backend

---

## About this guide

Read this guide if

- you'd like to know **how your plugin can persist new non-analytics data**
- you'd like to know **what information is stored when Piwik stores analytics data, log data and miscellaneous data**
- you'd like to know **how Piwik uses MySQL to persist data**

### Guide assumptions

This guide assumes that you:

- can write PHP and SQL code
- and understand how relational databases work in general and MySQL in particular.

## What gets persisted

Piwik persists two main types of data: log data and archive data. **Log data** is everything that Piwik tracks and **archive data** is analytics data that is cached.

Piwik also persists other simpler forms of data including information about each:

- website being tracked,
- user,
- goal,
- and [option](#).

This guide describes exactly what information this data consists of and exactly how the MySQL backend persists it.

*Note: Piwik uses PHP arrays to hold data that will be persisted. When we describe what information is in each persisted entity, we list properties by the string name used to store the property in the entity array.*

# Log Data Persistence

There are four types of log data, **visits**, **action types**, **conversions** and **ecommerce items**. All log data is persisted in a similar way: new data is constantly added to the set at high volume and updates are non-existent (except for **visits**).

**Visit** data is updated while visits are active. So until a visit ends it is possible that Piwik will try and update it.

Log data is read when calculating analytics data and old data will sometimes be deleted (via the [data purging feature](#)).

Backends must ensure that inserting new log data is as fast as possible and aggregating log data is not too slow (though obviously, faster is better).

## Visits

Each visit contains the following information:

- **'idsite'**

: the ID of the website it was tracked for

- **'idvisitor'**

: a visitor ID (an 8 byte binary string)

- **'visitor\_localtime'**

: the visit datetime in the visitor's time of day

- **'visitor\_returning'**

: whether the visit is the first visit for this visitor or not

- **'visitor\_count\_visits'**

: the number of visits the visitor has made up to this one

- **'visitor\_days\_since\_last'**

: the number of days since this visitor's last visit (if any)

- **'visitor\_days\_since\_order'**

: the number of days since this visitor's last order (if any)

- `'visitor_days_since_first'`

: the number of days since this visitors' first visit

- `'visit_first_action_time'`

: the datetime of the visit's first action

- `'visit_last_action_time'`

: the datetime of the visit's last action

- `'visit_exit_idaction_url'`

: the ID of the URL action type of the visit's last action

- `'visit_exit_idaction_name'`

: the ID of the page title action type of the visit's last action

- `'visit_entry_idaction_url'`

: the ID of the URL action type of the visit's first action

- `'visit_entry_idaction_name'`

: the ID of the page title action type of this visit's first action

- `'visit_total_actions'`

: the count of actions performed during this visit

- `'visit_total_searches'`

: the count of site searches performed during this visit

- `'visit_total_events'`

: the count of custom events performed during this visit

- `'visit_total_time'`

: the total elapsed time of the visit

- **'visit\_goal\_converted'**

: whether this visit converted a goal or not

- **'visit\_goal\_buyer'**

: whether the visitor ordered something during this visit or not

- **'referer\_type'**

: the type of this visitor'sreferrer. Can be one of the following values:

- [Common::REFERRER\\_TYPE\\_DIRECT\\_ENTRY](#): If set to this value, other

- **'referer\_...'**

fields have no meaning.

- [Common::REFERRER\\_TYPE\\_SEARCH\\_ENGINE](#): If set to this value,

- **'referer\_url'**

is the url of the search engine and

- **'referer\_keyword'**

is the keyword used (if we can find it). TODO: for search engine, will referer\_name be set? (same for website)

- [Common::REFERRER\\_TYPE\\_WEBSITE](#): If set to this value,

- **'referer\_url'**

is the url of the website.

- [Common::REFERRER\\_TYPE\\_CAMPAIGN](#): If set to this value,

- **'referer\_name'**

is the name of the campaign. TODO: double check campaign info

- **'referer\_name'**

: referrer name; its meaning depends on the specific referrer type

- **'referer\_url'**

: the referrer URL; its meaning depends on the specific referrer type

- **'referer\_keyword'**

: the keyword used if a search engine was the referrer

- **'config\_id'**

: a hash of all the visit's configuration options, including the OS, browser name, browser version, browser language, IP address and all browser plugin information

- **'config\_os'**

: a short string identifying the operating system used to make this visit.

See [UserAgentParser](#) for more info

- **'config\_browser\_name'**

: a short string identifying the browser used to make this visit.

See [UserAgentParser](#) for more info (TODO: what about devicesdetection?)

- **'config\_browser\_version'**

: a string identifying the version of the browser used to make this visit

- **'config\_resolution'**

: a string identifying the screen resolution the visitor used to make this visit (eg,

**'1024x768'**

)

- **'config\_pdf'**

: whether the visitor's browser can view PDF files or not

- **'config\_flash'**

: whether the visitor's browser can view flash files or not

- **'config\_java'**

: whether the visitor's browser can run Java or not

- **'config\_director'**

: TODO

- **'config\_quicktime'**

: whether the visitor's browser uses quicktime to play media files or not

- **'config\_realplayer'**

: whether the visitor's browser can play realplayer media files or not

- **'config\_windowsmedia'**

: whether the visitor's browser uses windows media player to play media files

- **'config\_gears'**

: TODO

- **'config\_silverlight'**

: whether the visitor's browser can run silverlight programs or not

- **'config\_cookie'**

: whether the visitor's browser has cookies enabled or not

- **'location\_ip'**

: the IP address of the computer that the visit was made from. can be[anonymized](#)

- **'location\_browser\_lang'**

: a string describing the language used in the visitor's browser

- **'location\_country'**

: a two character string describing the country the visitor was located in while visiting the site. set by the [UserCountry](#) plugin.

- **'location\_region'**

: a two character string describing the region of the country the visitor was in. set by the [UserCountry](#) plugin.

- '`location_city`'

: a string naming the city the visitor was in while visiting the site. set by the [UserCountry](#) plugin.

- '`location_latitude`'

: the latitude of the visitor while he/she visited the site. set by the [UserCountry](#) plugin.

- '`location_longitude`'

: the longitude of the visitor while he/she visited the site. set by the [UserCountry](#) plugin.

- '`custom_var_k1`'

: the custom variable name of the visit in the first slot for visit custom variables

- '`custom_var_v1`'

: the custom variable value of the visit in the first slot for visit custom variables

- '`custom_var_k2`'

: the custom variable name of the visit in the second slot for visit custom variables

- '`custom_var_v2`'

: the custom variable value of the visit in the second slot for visit custom variables

- '`custom_var_k3`'

: the custom variable name of the visit in the third slot for visit custom variables

- '`custom_var_v3`'

: the custom variable value of the visit in the third slot for visit custom variables

- '`custom_var_k4`'

: the custom variable name of the visit in the fourth slot for visit custom variables

- '`custom_var_v4`'

: the custom variable value of the visit in the fourth slot for visit custom variables

- `'custom_var_k5'`

: the custom variable name of the visit in the fifth slot for visit custom variables

- `'custom_var_v5'`

: the custom variable value of the visit in the fifth slot for visit custom variables

Some plugins, such as the [Provider](#) plugin, will add new information to visits.

## Visit Actions

Visits also contain a list of actions, one for each action the visitor makes during the visit.

Visit actions contain the following information:

- `'server_time'`

: the datetime the action was tracked in the server's timezone

- `'idaction_url'`

: the ID of the URL action type for this action

- `'idaction_url_ref'`

: the ID of the URL action type for the previous action in the visit

- `'idaction_name'`

: the ID of the page title action type for this action

- `'idaction_name_ref'`

: the ID of the page title action type for the previous action in the visit

- `'time_spent_ref_action'`

: the amount of time spent doing the previous action

- `'custom_var_k1'`

: the custom variable name of the first slot for page custom variables

- `'custom_var_v1'`

: the custom variable value of the first slot for page custom variables

- '`custom_var_k2`'

: the custom variable name of the second slot for page custom variables

- '`custom_var_v2`'

: the custom variable value of the second slot for page custom variables

- '`custom_var_k3`'

: the custom variable name of the third slot for page custom variables

- '`custom_var_v3`'

: the custom variable value of the third slot for page custom variables

- '`custom_var_k4`'

: the custom variable name of the fourth slot for page custom variables

- '`custom_var_v4`'

: the custom variable value of the fourth slot for page custom variables

- '`custom_var_k5`'

: the custom variable name of the slot for page custom variables

- '`custom_var_v5`'

: the custom variable value of the slot for page custom variables

- '`custom_float`'

: an unspecified float field, usually used to hold the time it took the server to serve this action

TODO: custom events when finished TODO: replace page title action type w/ action name?  
need to describe somewhere what an 'action name' is.

## Action Types

Action types, such as a specific URL or page title, are analyzed as well as visits. Such analysis can lead to an understanding of, for example, which pages are more relevant to visitors than others.

When Piwik encounters a new action type, a new action type entity is persisted.

Action types contain the following information:

- **'name'**

: a string describing the action type. Can be a URL, a page title, campaign name or anything else. The meaning is determined by the **type** field.

- **'hash'**

: a hash value calculated using the name.

- **'type'**

: the action type's category. Can be one of the following values:

- [Action::TYPE\\_PAGE\\_URL](#): the action type is a URL to a page on the website being tracked.
- [Action::TYPE\\_OUTLINK](#): the action type is a URL of a link on the website being tracked. A visitor clicked it.
- [Action::TYPE\\_DOWNLOAD](#): the action type is a URL of a file that was downloaded from the website being tracked.
- [Action::TYPE\\_PAGE\\_TITLE](#): the action type is the page title of a page on the website being tracked.
- [Action::TYPE\\_ECOMMERCE\\_ITEM\\_SKU](#): the action type is the SKU of an ecommerce item that is sold on the site.
- [Action::TYPE\\_ECOMMERCE\\_ITEM\\_NAME](#): the action type is the name of an ecommerce item that is sold on the site.
- [Action::TYPE\\_ECOMMERCE\\_ITEM\\_CATEGORY](#): the action type is the name of an ecommerce item category that is used on the site.
- [Action::TYPE\\_SITE\\_SEARCH](#): the action type is a site search action.

- **'url\_prefix'**

: if the name is a URL this refers to the prefix of the URL. The prefix is removed from actual URLs so the protocol and **www**. parts of a URL are ignored during analysis. Can be the following values:

- **0**

:

- **'http://'**

- **1**

```
:  
  'http://www.'  
o 2  
  
:  
  'https://'  
o 3  
  
:  
  'https://www.'
```

## Conversions

When a visit action is tracked that matches a goal's conversion parameters, a conversion is created and persisted. A conversion is a tally that counts a desired action that one of your visitors took. Piwik will analyze these tallies in conjunction with the actions that caused them help users understand how to make their visitors take more desired actions.

A conversion consists of the following information:

- **'idvisit'**  
: the ID of the visit that caused this conversion
- **'idsite'**  
: the ID of the site this conversion is for
- **'idvisitor'**  
: the ID of the visitor that caused this conversion
- **'server\_time'**  
: the datetime of the conversion in the server's timezone
- **'idaction\_url'**  
: the ID of the URL action type of the visit action that caused this conversion
- **'idlink\_va'**  
: the ID of the specific visit action that resulted in this conversion

- **'referer\_visit\_server\_date'**

: TODO: what is this? tied to \_refts query parameter

- **'url'**

: the URL that caused this conversion to be tracked

- **'idgoal'**

: the ID of the goal this conversion is for

- **'idorder'**

: if this conversion is for an ecommerce order or abandoned cart, this will be the order's ID

- **'items'**

: if this conversion is for an ecommerce order or abandoned cart, this will be the number of items in the order/cart

- **'revenue'**

: if this conversion is for an ecommerce order or abandoned cart, this is the total revenue generated by the order

- **'revenue\_subtotal'**

: if this conversion is for an ecommerce order or abandoned cart, this is the total cost of the items in the order/cart

- **'revenue\_tax'**

: if this conversion is for an ecommerce order or abandoned cart, this is the total tax applied to the items in the order/cart

- **'revenue\_shipping'**

: if this conversion is for an ecommerce order or abandoned cart, this is the total cost of shipping

- **'revenue\_discount'**

: if this conversion is for an ecommerce order or abandoned cart, this is the total discount applied to the order

# Ecommerce items (aka, conversion items)

An ecommerce item is an item that was sold in an ecommerce order or abandoned in an abandoned cart. They consist of the following information:

- **'server\_time'**

: TODO: time of visit action? or time ecommerce item was added?

- **'idorder'**

: the ID of the order that this ecommerce item is a part of

- **'idaction\_sku'**

: the ID of the action type entity that contains the item's SKU

- **'idaction\_name'**

: the ID of the action type entity that contains the ecommerce item's name

- **'idaction\_category'**

: the ID of an action type entity that contains a category for this ecommerce item

- **'idaction\_category2'**

: the ID of an action type entity that contains a category for this ecommerce item

- **'idaction\_category3'**

: the ID of an action type entity that contains a category for this ecommerce item

- **'idaction\_category4'**

: the ID of an action type entity that contains a category for this ecommerce item

- **'idaction\_category5'**

: the ID of an action type entity that contains a category for this ecommerce item

- **'price'**

: the price of this individual ecommerce item

- **'quantity'**

: the amount of this item that were present in the associated ecommerce order

- **'deleted'**

: whether this item was removed from the order or not

TODO: this action type stuff is rather confusing. needs to be described better or refactored.  
are they actually a type of action a user can take? if so, why is SKU an idaction? and why  
url + page title?

## Archive Data Persistence

Archive data consists of **metrics** and **reports**. Metrics are numeric values and are stored as such. Reports are stored in [DataTable](#) instances and persisted as compressed binary strings.

Archive data is associated with the website ID, period and segment it is for along with the data's identifying name. All archive data will be queried many times by this information. Currently, the segment is hashed and attached to the end of the metric name.

Archive data is also persisted with the current date & time so it is possible to know how old some data is.

All archive data will contain the following information:

- **'idarchive'**

: An ID that is shared with all pieces of archive data that were archived with the same website ID, period and segment.

- **'name'**

: The name of the report or metric. If a segment is used, a hash of the segment is appended to the name.

- **'idsite'**

: The ID of the website this archive data is for.

- **'date1'**

: The first date in the period this archive data is for.

- **'date2'**

: The last date in the period this archive data is for.

- '**period**'

: The type of period this archive data is for. Can be one of the following values:

- 1

: for **day** periods.

- 2

: for **week** periods.

- 3

: for **month** periods.

- 4

: for **year** periods.

- 5

: for **range** periods.

- '**ts\_archived**'

: The datetime the archive data was cached.

- '**value**'

: Either a numeric value (for a metric) or a binary string (for a report).

## Other Data Persistence

### Websites (aka sites)

**Site** entities contain information regarding a website whose visits are tracked. There won't be nearly as many of these as there are visits and archive data entries, but they will be queried often.

Every reporting request (either through the [Reporting API](#) or through Piwik's UI) will query one or more site entities. The tracker will only query site data if the [tracker cache](#) needs to be updated. For most tracking requests, site data will not be queried (thus resulting in greater performance for the tracker).

Site entities contain the following information:

- **'idsite'**

: the unique ID of the website.

- **'name'**

: the name of the website.

- **'main\_url'**

: the main URL visitors should use to access the website.

- **'ts\_created'**

: the date & time the site entity was persisted.

- **'ecommerce'**

:

**1**

if the site is an ecommerce site,

**0**

if not.

- **'sitesearch'**

:

**1**

if the site contains an internal search feature,

**0**

if not.

- **'sitesearch\_keyword\_parameters'**

: the query parameters the site uses to hold internal site search keywords. This is a comma separated list.

- **'sitesearch\_category\_parameters'**

: the query parameters the site uses to hold internal site search categories. This is a comma separated list.

- **'timezone'**

: the timezone of the website.

- **'currency'**

: the currency the website uses. Only valid if the site is an ecommerce site.

- **'excluded\_ips'**

: a comma separated list of IP addresses or IP address ranges. Visits that come from one of these IP addresses will not be tracked for this website.

- **'excluded\_parameters'**

: a comma separated list of query parameter names. These query parameters will be removed from page URLs before visits and actions are tracked.

- **'excluded\_user\_agents'**

: a comma separated list of strings. Visits with a user agent that contains one of these strings will not be tracked for this website.

- **'group'**

: TODO: is this implemented yet?

- **'keep\_url\_fragment'**

:

**1**

if the URL fragment (everything after the

**#**

) should be kept in the URL when tracking actions,

**0**

if not.

Site entities also contain a list of extra URLs that can be used to access the website. These are not stored within site entities themselves.

Site entity data access occurs primarily through the [Piwik\Site](#) class. Anything that cannot be queried through that class can be queried through the [SitesManager](#) core plugin.

## Goals

Each site has an optional list of goals. A goal is a desired action that a website visitor should take.

The following information is stored in a goal entity:

- **'idsite'**

: The ID of the website this goal belongs to.

- **'idgoal'**

: The ID for this goal (unique only among goals for this website).

- **'name'**

: The name of this goal.

- **'match\_attribute'**

: string describing what part of the request should be matched against when converting a goal. Can be one of the following values:

- **'manually'**

: the goal is converted by manual [conversion requests](#).

- **'url'**

: the goal is converted based on what the action URL contains.

- **'title'**

: the goal is converted based on what the action page title contains.

- **'file'**

: the goal is converted based on what the filename of a downloaded file

contains.

- o **'external\_website'**

: the goal is converted based on what the URL of an outlink contains.

- **'pattern'**

: the pattern to use when checking if a goal is converted.

- **'pattern\_type'**

: the type of pattern matching to use when checking if a goal is converted.

- o **'contains'**

: the goal is converted if the match attribute contains the pattern.

- o **'exact'**

: the goal is converted if the match attribute equals the pattern exactly.

- o **'regex'**

: the goal is converted if the match attribute is a regex match with the pattern.

- **'case\_sensitive'**

:

**1**

if the matching should be case sensitive,

**0**

if otherwise.

- **'allow\_multiple'**

:

**1**

if multiple conversions are allowed per visit,

0

if otherwise.

- **'revenue'**

: the amount of revenue a conversion generates (if any).

- **'deleted'**

:

1

if this goal was deleted by a Piwik user,

0

if not. TODO: why not remove goals? to keep conversions valid?

*Note: The ecommerce and abandoned cart goals are two special goals that have special IDs. Ecommerce websites automatically have these goals.*

## Users

User entities describe each Piwik user except the Super User. The following information is stored in a user entity:

- **'login'**

: the user's login handle.

- **'password'**

: a hash of the user's password.

- **'alias'**

: the user's alias if any. This value is displayed instead of the login handle when addressing the user in the UI.

- **'email'**

: the user's email address.

- **'token\_auth'**

: a user's [token auth](#).

- **'date\_registered'**

: the date the user data was persisted.

User data is read on every UI and [Reporting API](#) request. TODO: the tracker uses a token\_auth, does that mean it reads user data? or is that data cached? or is it just for the superuser so the config is used?

There is some user related information that is not stored directly in user entities. They are described below:

## User access

Users can be allowed and disallowed access to websites. Piwik persists each user's access level for each website they have access to. If they don't have access to a website, then no information regarding that user + website combination will be persisted.

An access level can be one of the following values:

- **'view'**

: The user has view access but cannot add goals or change any settings for the site.

- **'admin'**

: The user can view analytics data for the site and add goals or change settings for the site.

## User language choice

Piwik will also persist each user's language of choice. User logins will be associated with the name of the language (written in the chosen language as opposed to English). TODO: why not just associate it w/ the language code?

This association and the persistence logic is implemented by the [LanguagesManager](#) plugin.

## [Options](#)

Options are key-value pairs where the key is a string and the value is another string (possibly bigger and possibly binary). They are queried on every UI and [Reporting](#)

[API](#) request. The tracker will [cache](#) relevant option values and so will only query options when the cache needs updating.

Some options should be loaded on every non-tracking request. These options have a **specialautoload** property set to

1

## The Database Logging Backend

Piwik includes a [logging utility](#) that can be used to aid development or troubleshoot live Piwik installs. The utility can output log messages to multiple backends, including the database.

Every log entry contains the following information (all of which is persisted):

- **'tag'**

: A string used to categorize the log entry. This will either be the name of the plugin that logged a message or, if it cannot be found, the name of the class.

- **'timestamp'**

: When the log entry was made.

- **'level'**

: The log level (as a string). Describes the severity of the entry. See [Piwik\Log](#) for a list of levels.

- **'message'**

: The log entry's message.

## Plugin Persistence

Plugins can provide persistence for new data if they need to. At the moment, since MySQL is the only supported backend, this means directly adding and using new tables.

To add new tables to Piwik's MySQL database, execute a

`CREATE TABLE`

statement in the plugin descriptor's [install](#) method. For example:

`use Piwik\Db;`

```

public class MyPlugin extends \Piwik\Plugin
{
    // ...

    public function install()
    {
        try {
            $sql = "CREATE TABLE " . Common::prefixTable('mynewtable') . " (
                mykey VARCHAR( 10 ) NOT NULL ,
                mydata VARCHAR( 100 ) NOT NULL ,
                PRIMARY KEY ( mykey )
            ) DEFAULT CHARSET=utf8 ";
            Db::exec($sql);
        } catch (Exception $e) {
            // ignore error if table already exists (1050 code is for 'table already exists')
            if (!Db::get()->isErrNo($e, '1050')) {
                throw $e;
            }
        }
    }

    // ...
}

```

Plugins should also clean up after themselves by dropping the tables in the [uninstall](#) method:

```

use Piwik\Db;

public class MyPlugin extends \Piwik\Plugin

```

```

{

    // ...


    public function install()
    {

        Db::dropTables(Common::prefixTable('mynewtable'));

    }

    // ...
}


```

**Note:** New tables should be appropriately [prefixed](#).

## Augmenting existing tables

Plugins can also augment existing tables. If, for example, a plugin wanted to track extra visit information, the plugin could add columns to the log data tables to have a place to put this new data. This would also be done in the [install](#) method:

```

use Piwik\Db;

public class MyPlugin extends \Piwik\Plugin
{


    public function install()
    {
        try {
            $q1 = "ALTER TABLE `".Common::prefixTable("log_visit")."` "
                ADD `mynewdata` VARCHAR( 100 ) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL AFTER `config_os`,";
            Db::exec($q1);
        }
    }
}


```

```

} catch (Exception $e) {
    // ignore column already exists error
    if (!Db::get()->isErrNo($e, '1060')) {
        throw $e;
    }
}

}

// ...
}

```

Plugins should remove the column in the [uninstall](#) method, **unless doing so take very long time**. Since log tables can have millions and even billions of entries, removing columns from these tables when a plugin is uninstalled would be a bad idea.

TODO: this seems it will be a problem in the long run. must be some way to clear away unused data, even if the user has to say specifically they want to get rid of it.

## The MySQL Backend

This section lists each MySQL table used to store the data described above and details everything we did to them to make Piwik run as fast as possible.

*Note: All table names in the MySQL database are prefixed with the value in the*

*[database] tables\_prefix*

*INI config.* This is to ensure that non-piwik tables do not get overwritten or used by accident. It also makes it possible to store multiple instances of Piwik in one database.

## Log Data Tables

### log\_visit

This table stores [Visit entities](#).

The

`CREATE TABLE`

SQL for this table is:

`CREATE TABLE log_visit (`

```
idvisit INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,  
idsite INTEGER(10) UNSIGNED NOT NULL,  
idvisitor BINARY(8) NOT NULL,  
visitor_localtime TIME NOT NULL,  
visitor_returning TINYINT(1) NOT NULL,  
visitor_count_visits SMALLINT(5) UNSIGNED NOT NULL,  
visitor_days_since_last SMALLINT(5) UNSIGNED NOT NULL,  
visitor_days_since_order SMALLINT(5) UNSIGNED NOT NULL,  
visitor_days_since_first SMALLINT(5) UNSIGNED NOT NULL,  
visit_first_action_time DATETIME NOT NULL,  
visit_last_action_time DATETIME NOT NULL,  
visit_exit_idaction_url INTEGER(11) UNSIGNED NULL DEFAULT  
0,  
visit_exit_idaction_name INTEGER(11) UNSIGNED NOT NULL,  
visit_entry_idaction_url INTEGER(11) UNSIGNED NOT NULL,  
visit_entry_idaction_name INTEGER(11) UNSIGNED NOT NULL,  
visit_total_actions SMALLINT(5) UNSIGNED NOT NULL,  
visit_total_searches SMALLINT(5) UNSIGNED NOT NULL,  
visit_total_events SMALLINT(5) UNSIGNED NOT NULL,  
visit_total_time SMALLINT(5) UNSIGNED NOT NULL,  
visit_goalConverted TINYINT(1) NOT NULL,  
visit_goalBuyer TINYINT(1) NOT NULL,  
referer_type TINYINT(1) UNSIGNED NULL,  
referer_name VARCHAR(70) NULL,  
referer_url TEXT NOT NULL,  
referer_keyword VARCHAR(255) NULL,  
config_id BINARY(8) NOT NULL,  
config_os CHAR(3) NOT NULL,  
config_browser_name VARCHAR(10) NOT NULL,  
config_browser_version VARCHAR(20) NOT NULL,
```

```
config_resolution VARCHAR(9) NOT NULL,  
config_pdf TINYINT(1) NOT NULL,  
config_flash TINYINT(1) NOT NULL,  
config_java TINYINT(1) NOT NULL,  
config_director TINYINT(1) NOT NULL,  
config_quicktime TINYINT(1) NOT NULL,  
config_realplayer TINYINT(1) NOT NULL,  
config_windowsmedia TINYINT(1) NOT NULL,  
config_gears TINYINT(1) NOT NULL,  
config_silverlight TINYINT(1) NOT NULL,  
config_cookie TINYINT(1) NOT NULL,  
location_ip VARBINARY(16) NOT NULL,  
location_browser_lang VARCHAR(20) NOT NULL,  
location_country CHAR(3) NOT NULL,  
location_region char(2) DEFAULT NULL,  
location_city varchar(255) DEFAULT NULL,  
location_latitude float(10, 6) DEFAULT NULL,  
location_longitude float(10, 6) DEFAULT NULL,  
custom_var_k1 VARCHAR(200) DEFAULT NULL,  
custom_var_v1 VARCHAR(200) DEFAULT NULL,  
custom_var_k2 VARCHAR(200) DEFAULT NULL,  
custom_var_v2 VARCHAR(200) DEFAULT NULL,  
custom_var_k3 VARCHAR(200) DEFAULT NULL,  
custom_var_v3 VARCHAR(200) DEFAULT NULL,  
custom_var_k4 VARCHAR(200) DEFAULT NULL,  
custom_var_v4 VARCHAR(200) DEFAULT NULL,  
custom_var_k5 VARCHAR(200) DEFAULT NULL,  
custom_var_v5 VARCHAR(200) DEFAULT NULL,  
PRIMARY KEY(idvisit),
```

```

        INDEX index_idsite_config_datetime (idsite, config_id, visit_last_action_time),
        INDEX index_idsite_datetime (idsite, visit_last_action_time),
        INDEX index_idsite_idvisitor (idsite, idvisitor)
) DEFAULT CHARSET=utf8;

```

The **index\_idsite\_config\_datetime** index is used when trying to recognize returning visitors.

The **index\_idsite\_datetime** index is used when aggregating visits. Since log aggregation occurs only for individual day periods, this index helps Piwik find the visits for a website and period quickly. Without it, log aggregation would require a table scan through the entire log\_visit table.

TODO: what is the index\_idsite\_idvisitor index used for? Live + visitor profile?

## log\_link\_visit\_action

This table stores [Visit Action entities](#).

The

`CREATE TABLE`

SQL for this table is:

```

CREATE TABLE log_link_visit_action (
    idlink_va INTEGER(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    idsite int(10) UNSIGNED NOT NULL,
    idvisitor BINARY(8) NOT NULL,
    server_time DATETIME NOT NULL,
    idvisit INTEGER(10) UNSIGNED NOT NULL,
    idaction_url INTEGER(10) UNSIGNED DEFAULT NULL,
    idaction_url_ref INTEGER(10) UNSIGNED NULL DEFAULT 0,
    idaction_name INTEGER(10) UNSIGNED,
    idaction_name_ref INTEGER(10) UNSIGNED NOT NULL,
    idaction_event_category INTEGER(10) UNSIGNED DEFAULT NULL,
    idaction_event_action INTEGER(10) UNSIGNED DEFAULT NULL,
    time_spent_ref_action INTEGER(10) UNSIGNED NOT NULL,

```

```

custom_var_k1 VARCHAR(200) DEFAULT NULL,
custom_var_v1 VARCHAR(200) DEFAULT NULL,
custom_var_k2 VARCHAR(200) DEFAULT NULL,
custom_var_v2 VARCHAR(200) DEFAULT NULL,
custom_var_k3 VARCHAR(200) DEFAULT NULL,
custom_var_v3 VARCHAR(200) DEFAULT NULL,
custom_var_k4 VARCHAR(200) DEFAULT NULL,
custom_var_v4 VARCHAR(200) DEFAULT NULL,
custom_var_k5 VARCHAR(200) DEFAULT NULL,
custom_var_v5 VARCHAR(200) DEFAULT NULL,
custom_float FLOAT NULL DEFAULT NULL,
PRIMARY KEY(idlink_va),
INDEX index_idvisit(idvisit),
INDEX index_idsite_servertime (idsite, server_time )
) DEFAULT CHARSET=utf8

```

The

`idsite`

and

`idvisitor`

columns are copied from the visit action's associated visit in order to avoid having to join the `log_visit` table in some cases.

The `index_idvisit` index allows Piwik to quickly query the visit actions for a visit.

The `index_idsite_servertime` index is used when aggregating visit actions. It allows quick access to the visit actions that were tracked for a specific website during a specific period and lets us avoid a table scan through the whole table.

## log\_action

This table stores [Action Type entities](#).

The

`CREATE TABLE`

SQL for this table is:

```

CREATE TABLE log_action (
    idaction INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,

```

```

    name TEXT,
    hash INTEGER(10) UNSIGNED NOT NULL,
    type TINYINT UNSIGNED NULL,
    url_prefix TINYINT(2) NULL,
    PRIMARY KEY(idaction),
    INDEX index_type_hash (type, hash)
) DEFAULT CHARSET=utf8

```

The **index\_type\_hash** index is used during tracking to find existing action types.

## log\_conversion

This table stores [Conversion entities](#).

The

`CREATE TABLE`

SQL for this table is:

```

CREATE TABLE `log_conversion` (
    idvisit int(10) unsigned NOT NULL,
    idsite int(10) unsigned NOT NULL,
    idvisitor BINARY(8) NOT NULL,
    server_time datetime NOT NULL,
    idaction_url int(11) default NULL,
    idlink_va int(11) default NULL,
    referer_visit_server_date date default NULL,
    referer_type int(10) unsigned default NULL,
    referer_name varchar(70) default NULL,
    referer_keyword varchar(255) default NULL,
    visitor_returning tinyint(1) NOT NULL,
    visitor_count_visits SMALLINT(5) UNSIGNED NOT NULL,
    visitor_days_since_first SMALLINT(5) UNSIGNED NOT NULL,
    visitor_days_since_order SMALLINT(5) UNSIGNED NOT NULL,
    location_country char(3) NOT NULL,
    location_region char(2) DEFAULT NULL,

```

```

location_city varchar(255) DEFAULT NULL,
location_latitude float(10, 6) DEFAULT NULL,
location_longitude float(10, 6) DEFAULT NULL,
url text NOT NULL,
idgoal int(10) NOT NULL,
buster int unsigned NOT NULL,
idorder varchar(100) default NULL,
items SMALLINT UNSIGNED DEFAULT NULL,
revenue float default NULL,
revenue_subtotal float default NULL,
revenue_tax float default NULL,
revenue_shipping float default NULL,
revenue_discount float default NULL,
custom_var_k1 VARCHAR(200) DEFAULT NULL,
custom_var_v1 VARCHAR(200) DEFAULT NULL,
custom_var_k2 VARCHAR(200) DEFAULT NULL,
custom_var_v2 VARCHAR(200) DEFAULT NULL,
custom_var_k3 VARCHAR(200) DEFAULT NULL,
custom_var_v3 VARCHAR(200) DEFAULT NULL,
custom_var_k4 VARCHAR(200) DEFAULT NULL,
custom_var_v4 VARCHAR(200) DEFAULT NULL,
custom_var_k5 VARCHAR(200) DEFAULT NULL,
custom_var_v5 VARCHAR(200) DEFAULT NULL,
PRIMARY KEY (idvisit, idgoal, buster),
UNIQUE KEY unique_idsite_idorder (idsite, idorder),
INDEX index_idsite_datetime (idsite, server_time)
) DEFAULT CHARSET=utf8

```

All extra information stored in this table that is not listed for the conversion entity above is replicated from the Visit entity this conversion is for. This allows us to avoid joining the log\_visit table in certain cases.

The **index\_idsite\_datetime** index is used when aggregating conversions. It allows quick access to the conversions that were tracked for a specific website during a specific period and lets us avoid a table scan through the entire table.

## log\_conversion\_item

This table stores [Ecommerce Item entities](#).

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE `log_conversion_item` (
    idsite int(10) UNSIGNED NOT NULL,
    idvisitor BINARY(8) NOT NULL,
    server_time DATETIME NOT NULL,
    idvisit INTEGER(10) UNSIGNED NOT NULL,
    idorder varchar(100) NOT NULL,
    idaction_sku INTEGER(10) UNSIGNED NOT NULL,
    idaction_name INTEGER(10) UNSIGNED NOT NULL,
    idaction_category INTEGER(10) UNSIGNED NOT NULL,
    idaction_category2 INTEGER(10) UNSIGNED NOT NULL,
    idaction_category3 INTEGER(10) UNSIGNED NOT NULL,
    idaction_category4 INTEGER(10) UNSIGNED NOT NULL,
    idaction_category5 INTEGER(10) UNSIGNED NOT NULL,
    price FLOAT NOT NULL,
    quantity INTEGER(10) UNSIGNED NOT NULL,
    deleted TINYINT(1) UNSIGNED NOT NULL,
    PRIMARY KEY(idvisit, idorder, idaction_sku),
    INDEX index_idsite_servertime ( idsite, server_time )
) DEFAULT CHARSET=utf8
```

The

```
idsite  
,  
idvisitor  
,  
server_time  
and  
idvisit
```

columns are copied from the Conversion entity this Ecommerce Item belongs to. They are copied so we can aggregate Ecommerce Items without having to join other tables.

The **index\_idsite\_servertime** index is used when aggregating ecommerce items. It allows quick access to the items that were tracked for a specific website and during a specific period and lets us avoid a table scan through the entire table.

## Archive Tables

In the MySQL backend archive data is partitioned by the month the archive data is for. So reports that aggregate visits from January, 2012 will be held in a different table from reports that aggregate visits from February 2012.

Piwik creates two types of archive tables, one for each type of archive data.

The **archive\_numeric** tables store metric data and the **archive\_blob** tables store report data.

Archive tables are created dynamically. When Piwik needs to query or insert archive data for a certain month and it cannot find the table that holds this data, the table is created.

The year and month of an archive table is appended as the suffix to the name. So the **archive\_numeric** table for January, 2012 would have the name **archive\_numeric\_2012\_01**.

### archive\_numeric

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE archive_numeric_YYYY_MM (  
    idarchive INTEGER UNSIGNED NOT NULL,  
    name VARCHAR(255) NOT NULL,  
    idsite INTEGER UNSIGNED NULL,  
    date1 DATE NULL,  
    date2 DATE NULL,
```

```

    period TINYINT UNSIGNED NULL,
    ts_archived DATETIME NULL,
    value DOUBLE NULL,
    PRIMARY KEY(idarchive, name),
    INDEX index_idsite_dates_period(idsite, date1, date2, period, ts_archived),
    INDEX index_period_archived(period, ts_archived)
) DEFAULT CHARSET=utf8

```

The **index\_idsite\_dates\_period** index is used when querying archive data. It lets Piwik quickly query archive data for any site and period, and for data that was archived past a certain date-time.

The **index\_period\_archived** index is used when [purging archive data](#). It allows Piwik to quickly find archive data for a specific period that is old enough to be purged.

## archive\_blob

The

**CREATE TABLE**

SQL for this table is:

```

CREATE TABLE archive_blob (
    idarchive INTEGER UNSIGNED NOT NULL,
    name VARCHAR(255) NOT NULL,
    idsite INTEGER UNSIGNED NULL,
    date1 DATE NULL,
    date2 DATE NULL,
    period TINYINT UNSIGNED NULL,
    ts_archived DATETIME NULL,
    value MEDIUMBLOB NULL,
    PRIMARY KEY(idarchive, name),
    INDEX index_period_archived(period, ts_archived)
) DEFAULT CHARSET=utf8

```

The **index\_period\_archived** index is used in the same way as the one in **archive\_numerictables**.

**archive\_blob** tables do not have an index that makes it fast to query for rows by site, period and archived date. This is because they should not be queried this way. Instead, the **archive\_numerictable** should be queried and the

**idarchive**

values saved. These values can be used to query data in the **archive\_blob** table.

## Other Tables

**site**

This table stores [Website](#) entities.

The

**CREATE TABLE**

SQL for this table is:

```
CREATE TABLE site (
    idsite INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(90) NOT NULL,
    main_url VARCHAR(255) NOT NULL,
    ts_created TIMESTAMP NULL,
    ecommerce TINYINT DEFAULT 0,
    sitesearch TINYINT DEFAULT 1,
    sitesearch_keyword_parameters TEXT NOT NULL,
    sitesearch_category_parameters TEXT NOT NULL,
    timezone VARCHAR( 50 ) NOT NULL,
    currency CHAR( 3 ) NOT NULL,
    excluded_ips TEXT NOT NULL,
    excluded_parameters TEXT NOT NULL,
    excluded_user_agents TEXT NOT NULL,
    `group` VARCHAR(250) NOT NULL,
    keep_url_fragment TINYINT NOT NULL DEFAULT 0,
    PRIMARY KEY(idsite)
) DEFAULT CHARSET=utf8
```

site\_url

This table stores extra URLs for [Website](#) entities.

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE site_url (
    idsite INTEGER(10) UNSIGNED NOT NULL,
    url VARCHAR(255) NOT NULL,
    PRIMARY KEY(idsite, url)
) DEFAULT CHARSET=utf8
```

goal

This table stores [Goal](#) entities.

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE `goal` (
    `idsite` int(11) NOT NULL,
    `idgoal` int(11) NOT NULL,
    `name` varchar(50) NOT NULL,
    `match_attribute` varchar(20) NOT NULL,
    `pattern` varchar(255) NOT NULL,
    `pattern_type` varchar(10) NOT NULL,
    `case_sensitive` tinyint(4) NOT NULL,
    `allow_multiple` tinyint(4) NOT NULL,
    `revenue` float NOT NULL,
    `deleted` tinyint(4) NOT NULL default '0',
    PRIMARY KEY  (`idsite`,`idgoal`)
) DEFAULT CHARSET=utf8
```

## users

This table stores [User](#) entities.

The

```
CREATE TABLE
SQL for this table is:
CREATE TABLE user (
    login VARCHAR(100) NOT NULL,
    password CHAR(32) NOT NULL,
    alias VARCHAR(45) NOT NULL,
    email VARCHAR(100) NOT NULL,
    token_auth CHAR(32) NOT NULL,
    date_registered TIMESTAMP NULL,
    PRIMARY KEY(login),
    UNIQUE KEY uniq_keytoken(token_auth)
) DEFAULT CHARSET=utf8
```

access

This table stores [User Access information](#).

The

```
CREATE TABLE
SQL for this table is:
CREATE TABLE access (
    login VARCHAR(100) NOT NULL,
    idsite INTEGER UNSIGNED NOT NULL,
    access VARCHAR(10) NULL,
    PRIMARY KEY(login, idsite)
) DEFAULT CHARSET=utf8
```

user\_language

This table stores [User Language Choice information](#).

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE user_language (
    login VARCHAR( 100 ) NOT NULL ,
    language VARCHAR( 10 ) NOT NULL ,
    PRIMARY KEY ( login )
) DEFAULT CHARSET=utf8
```

This table is created by the [LanguagesManager](#) plugin.

## option

This table stores [Option](#) data.

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE `option` (
    option_name VARCHAR( 255 ) NOT NULL,
    option_value LONGTEXT NOT NULL,
    autoload TINYINT NOT NULL DEFAULT '1',
    PRIMARY KEY ( option_name ),
    INDEX autoload( autoload )
) DEFAULT CHARSET=utf8
```

## logger\_message

This table is used by the database logging backend.

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE logger_message (
    idlogger_message INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    tag VARCHAR(50) NULL,
    timestamp TIMESTAMP NULL,
    level VARCHAR(16) NULL,
```

```
    message TEXT NULL,  
    PRIMARY KEY(idlogger_message)  
) DEFAULT CHARSET=utf8
```

## session

This table does not store entity data. It is used by Piwik to store session data (as an alternative to using file-based sessions).

The

```
CREATE TABLE
```

SQL for this table is:

```
CREATE TABLE session (  
    id CHAR(32) NOT NULL,  
    modified INTEGER,  
    lifetime INTEGER,  
    data TEXT,  
    PRIMARY KEY ( id )  
) DEFAULT CHARSET=utf8
```

## Other Backends

Currently the MySQL backend is the only available backend. The use of MySQL is scattered throughout Piwik, so at the moment it is also not possible to create other backends.

That being said, the use of other persistence solutions is on our TODO list. Piwik will eventually work with different relational databases and with different NoSQL solutions.

## Learn more

- To learn **how the tracker inserts log data** see our [All About Tracking](#) guide.
- To learn **how log data is aggregated** see our [All About Analytics](#) guide and take a look at the [LogAggregator](#) class.
- To learn **how archive data is cached** see our [All About Analytics](#) guide.
- To learn **about Piwik's logging utility** see this section in our [Getting started extending Piwik](#) guide.
- To learn **about database backed sessions** read [this part of Piwik's user guide](#).

# Piwik on the Command Line

---

## About this guide

### Read this guide if

- you'd like to know **how to use a specific command with Piwik's command line tool**
- you'd like to know **how your plugin can create a new command for the command line tool**
- you'd like to know more about **how the command line tool works**

### Guide assumptions

This guide assumes that you:

- can code in PHP,
- and have installed Piwik (if not read our [Getting Started](#) guide)

## The **console** tool

Piwik comes with a special command line tool that can be used to make development easier. The tool is located in Piwik's root directory and is called **console**. You can run it with the command:

```
./console help
```

or

```
php ./console help
```

### Libraries Used

The console app uses the [Symfony Console component](#). It would have been installed when you ran [composer.phar](#) while installing Piwik.

## Commands

The command line tool currently supports commands that generate empty plugins and plugin files, run git commands, watch Piwik's log output, run tests and deal with Piwik translations.

You can view the entire list of commands by running the following command:

```
./console list
```

To get more information about a single command (such as what arguments it takes), run the following command:

```
./console help <<command>>
```

where

```
<<command>>
```

should be replaced with the command you are interested in.

## Adding new commands

Plugins can extend the command line tool by creating their own commands. To create your own command, create a class that extends [ConsoleCommand](#) and expose it using the[Console.addCommands](#) event.

See the documentation for the class and event mentioned and see the docs for [SymfonyCommand](#)to learn more about how your command should be coded.

# Contributing to Piwik Core

---

## About this guide

### Read this guide if

- you've found a bug, fixed it and want to know how to get your fix accepted upstream
- you're interested in contributing changes to Piwik and *want to know where to start or want what the process is like*
- you'd like to know what Piwik developers consider to be good code

### Guide assumptions

This guide assumes that you:

- can code in PHP and JavaScript,
- can use the [git](#) source code management tool
- are familiar with [github](#),
- and have the necessary tools to contribute to Piwik (if not, see this section of our [Getting started extending Piwik](#) guide).

# Contribution Process

The contribution process starts with a bug you want to fix or an idea that you want to implement. *If you don't have one, feel free to pick an open ticket on [dev.piwik.org](#).*

Once you've decided on something, continue below.

## Getting a copy of Piwik to work on

Before you can start contributing you need to get setup with [git](#) & [github](#). First, get a [github account here](#). Then login and...

### Fork the Piwik repository

While logged in, visit [Piwik's github repo](#). In the upper right corner there will be a *Fork* button. Click it. Github will copy the repository and place the new forked copy in your account.

### Setup git

When committing, git will need to know your username & email. To set them, run the following commands (replacing

```
$username
```

with your github username and

```
$email
```

with the email address you told github about):

```
git config --global user.name $username
```

```
git config --global user.email $email
```

Also, if you have a favorite text editor you'd like to use when creating commit messages, you can configure git to launch it when you commit changes. For example:

```
git config --global core.editor my-text-editor
```

### Clone the forked repository

To be able to work on Piwik you'll need to have a local copy of your forked repository on your computer. To copy your forked repository, run the following command after replacing **myusername** with your github username:

```
git clone https://github.com/myusername/piwik piwik
```

This will copy the entire forked repository (including all history) to the *piwik* folder on your hard drive.

Now, we'll run one more command so git remembers the original Piwik repository in addition to your fork:

```
git remote add upstream https://github.com/piwik/piwik
```

This will save <https://github.com/piwik/piwik> as a remote and name it *upstream*.

## Configuring up PHP

Contributions should not generate PHP errors or warnings. Applying the following settings to your php.ini file will enable you to catch these errors:

```
display_errors = On  
error_reporting = E_ALL | E_STRICT
```

## Hacking Piwik

Now that you have a copy of the latest Piwik source code, you can start modifying it. For this section, we'll assume there's a bug that you found and want to fix.

### Create a new branch

Before you start coding, you should make sure to keep your changes separate from the master branch. This will make it much easier to track the changes specific to your feature since your changes won't be mixed with any new commits to the master branch from other developers.

We'll give our new branch a name, *bugfix*, that describes what we're doing so we can recognize it later. To add a new branch, run the following command:

```
git checkout -b bugfix
```

The checkout command will create a new branch if the *-b* option is supplied, otherwise it will try and load an existing branch.

### Work on feature

Once you've created a branch, you have a place to start working on the feature. There's nothing special about this part of the process, just fix the bug or finish the feature you're working on.

If you're working on something more complex than a bugfix, though, you may have the need to keep your new branch updated with changes from the main repository. Keeping your branch up to date is a two-three step process. First, on your **master** branch (remember to use the *checkout* command to switch branches), pull changes from the main Piwik repository, nicknamed *upstream*:

```
git pull upstream master
```

Then, on your new branch (**bugfix** for this tutorial), merge with **master**:

```
git merge master
```

If there are conflicts, git will list them and tell you to fix them and then commit. To fix commits, you can do it manually in a text editor (which is usually pretty simple), or you can let git launch a GUI by calling:

```
git mergetool
```

What git will launch depends on what diff-ing tools you have installed.

After you fix conflicts, you have to let git know the conflicts are gone and re-commit. To do that, you run *git add* on the files with conflicts then *git commit*. More on how to add files and commit in the next section.

## Save your changes

Now that you've finished the bug fix or new feature (or just part of it), it's time to save your changes. First, you want to commit them to your local clone of your github repository. To do that, run the following command:

```
git commit -a -m "Fixed a bug."
```

This will commit every modified file and use "*Fixed a bug.*" as the commit message. If you have new files to add, run the following command before committing:

```
git add /path/to/file
```

If you only want to commit some files and not every modified file, run the above command for every modified file you want to commit, then run:

```
git commit -m "..."
```

The *-a* option (which we used above) tells git to automatically commit every file that has been modified, not just the specific files that have been used with *git add*.

If you delete a file, git will notice it is deleted, but you still have to *git add* the file or use the *-a* option to commit the deletion.

## Publish your changes

Now that you've committed your changes locally, you need to make them visible on your github repository. This is done with one command:

```
git push origin bugfix
```

This command will push every change you made on the **bugfix** branch to the **origin** remote (which is your github repository).

You can also use this command to publish changes to your **master** branch by running:

```
git push origin master
```

## Create pull request & wait for someone to review the code

Your changes are published, which means you can now send a pull request to the main Piwik project. To do this, visit your forked repository in a browser, and select the **bugfix** branch in the branch selector (located right above the directory listing on the left side of the page). Then click the *Pull Request* button in the middle of the top of the page.

On this screen you'll be able to see exactly what changes you've made by looking at your commits and at what files have been changed. You can use this as an opportunity to review your own code if you like.

Once you're ready to create the pull request, write a description of the pull request and any notes that are important for the person who will review your code, and then click *Send pull request*. This will create a new pull request which you will be able to find and view [here](#).

## Based on the review, make changes & push those changes

Once your pull request is public, someone will review it and leave comments regarding what should be changed. You can then make changes, commit them and then push them to your remote repository (**origin**) and they will automatically be shown in the pull request.

To see our coding standards for Piwik Core see [this section](#) below.

## Once your pull request has been merged, sync w/ master

Your pull request has been reviewed and deemed worthy: it has been merged with Piwik's **master** branch. Now, you need to sync your fork with the main repo. This is the same process that was described in [Work on Feature](#):

```
# first make sure you're on master
git checkout master
```

```
# now we'll pull in changes from 'upstream' the original Piwik  
repository  
  
git pull upstream master  
  
# then we'll push those changes to 'origin'  
git push origin master
```

And finally, now that the changes are part of Piwik, there's no need for your feature branch, so we'll delete it:

```
git branch -D bugfix
```

## Piwik Core Code Standards

The following are a list of guidelines and requirements for contributions to Piwik Core. Developers and teams interested in contributing should read through them before starting to contribute and before sending a pull request. ***Contributions that are not up to these standards will not be accepted.***

## General Considerations

- **Write clear, easily understandable code.** Is your code easy to read? Do you understand what each line and function does immediately after reading them? Code that does not do anything complicated should be this easy to understand. Complex code should have extra documentation that will aid new developers in understanding it.
- **Reuse as much code as possible.** Have you removed any redundancies in your code? Have you made sure your code does not replicate existing functionality? Try to reduce the amount of code you need to write for your contribution.
- **Write correct code.** Does your code handle all possible scenarios? Does your code handle all possible error conditions (including any corner cases)? Do existing tests pass for your contribution? Does your code generate any unwanted PHP errors or warnings? We do not want contributions that introduce new bugs.
- **Write efficient code.** Does your code scale well? What happens when there are hundreds of thousands of websites and billions of visits? Please note any potential performance issues and whether they would be easy to fix. We know how hard it can be to scale efficiently, but we would like for Piwik to be as fast as possible.
- **Follow our [security guidelines](#).** We do not allow any security vulnerabilities in contributions.

# Specific Considerations

- **Add new configuration settings for fixed values that may users may change.** Does your code use constants that users may want to change? They should be made configurable.
- **Use automated testing to test your PHP.** If you've written new PHP code, have you created unit and integration tests for them? Some code cannot be tested this way (such as code in controllers), but all code that could benefit from automated tests should have tests written for them. Read our [Automated Testing](#) guide to learn about Piwik's test suite.
- **Internationalize your text.** If your contribution has text, is it loaded from a language file? We want all text in Piwik Core to be available in as many languages as possible. To learn more about i18n in Piwik read our [Internationalization](#) guide.
- **Generate HTML in template files only.** Does your PHP code contain HTML? It shouldn't. All HTML generation should be handled by Twig templates.
- **If your contribution includes changes to the UI, make sure to do manual testing of all relevant areas of Piwik's UI.** See [this section](#) for details on how to do these tests.
- **If your contribution includes third-party components & libraries, make sure they include GPL compatible licenses.** Third-party components/libraries must be compatible with GPL v3 since this is the license used by Piwik.
- **Make sure your code follows the [PSR 0](#), [PSR 1](#) and [PSR 2](#) coding standards.**
- **Make sure your source code files are encoded in UTF8.**

TODO: where to put these? \* Several Piwik devs are using PhpStorm IDE. We have published our [customized PSR coding style xml file](#) for PhpStorm if you wish to reuse it. \* Files in Piwik's Git repository are stored using Linux EOL markers (LF).

# Coding Considerations

The following are a list of guidelines you should follow while writing code and architecting software:

## Include Path

Piwik does not set or depend on the include path. Plugins should rely on the autoloader for classes and use the absolute path convention for other files:

```
require_once PIWIK_INCLUDE_PATH . '/folder/script.php';
```

## Basic Clean Code recommendations

About classes:

- A class should follow the Single Responsibility Principle.
- Refactor classes and aim for files/classes that are at most 400 lines.
- Avoid classes with both public attributes and getters/setters. Choose to use getters and setters only when they make code easier to read.
- Add

**private**

keywords to class attributes when forgotten.

About methods and functions:

- A function should follow the Single Responsibility Principle: each function should do only one thing.
- Think about whether you can refactor the function body into smaller private methods.
- Aim for a maximum of 20-30 lines per method.
- Aim for maximum three function parameters.
- Extract the body of the

**try {}**

blocks into their own private method.

Keep the following principles (from Alan Shalloway) in mind while writing code: cohesion, loose coupling, no redundancy, encapsulation, testability, readability, and focus.

## Commenting

In order for new developers to get up to speed quickly and in order to lessen the amount of bugs Piwik will ever experience, the Piwik source code must be easy to understand. Comments, in addition to general code cleanliness, are important in achieving this goal.

Comments are a central part of professional coding. Comments can be divided into three categories: **documentary** (serving the purpose of documenting the evolution of code over time), **functional** (helping the co-ordination of the development effort inside the team) and **explanatory** (generating the software documentation for general use). All three categories are vital to the success of a software development project.

— from [The fine Art of Commenting](#)

For an example of a well commented Piwik class, see [Piwik Cookie](#).

Despite their importance, comments can sometimes cause information overload — or worse for out of date comments. Useless or inaccurate comments and autogenerated comments that add no value should be avoided. Rather than writing comments inside a function, it is better to write shorter functions that do only one thing and are named thoughtfully. A well refactored class made of small methods will be easy to read and will not need many comments.

## No duplication

**No duplication** is a basic and core principle of extreme programming and of writing good code in general. Write code "**Once, and only once**", i.e. Don't Repeat Yourself. Do not duplicate code.

## Debugging & logging

In Piwik you can easily log debug messages to files, the database or the screen by enabling logging in your config/config.ini.php file. In your code, you can then call

```
Piwik\Log::info($message);
```

to log new messages. You can also show the debug output on screen by appending &debug to the URL: this allows you to test and view debug messages only when needed, but still leave debug logging enabled. See the FAQ to [enable logging output in Piwik](#) and the various options.

Piwik also comes with a SQL profiler, which reports the time spent by each query, how many times they were called, the total time spent in Mysql vs php, etc. It makes it easier to check the performance and overhead of your new code. See the FAQ to see how enable [SQL profiling](#).

## Automated Tests

If you are fixing a small bug or simply modifying the UI, there is usually no need to write new tests. But if you are adding a new feature to Piwik, adding a new API method or modifying a core Archiving or Tracking algorithm, generally it is required to also submit new or updated Unit tests or Integration tests.

For more information about running or creating tests, read our [Automated Testing](#) guide.

When naming unit/integration tests, it helps to use the **Given, When, Then** convention for writing tests.

Tests are critical part of ensuring Piwik stays a stable and useful software platform. We aim to keep test code as clean as production code so it is easy to improve and maintain.

## Manual UI Testing

If you have modified the Javascript code, CSS, HTML, or PHP code that can affect the User Interface, you must manually test the UI to make sure everything still works. Read about each manual check and test in our [Manual UI Testing](#) guide.

If all UI tests pass, you have successfully made a change to the Piwik UI.

## Learn more

- To learn the basics of Piwik development in our [Getting started extending Piwik](#) guide.
- To see a list of available things you could work on see our [roadmaps](#)
- To learn more about Piwik's test suite see our [Automated Testing](#) guide.

# Core Team Workflow

---

## About this guide

This guide describes how we, the team of developers that makes changes to Piwik Core, operate and how others can participate in our work.

### Read this guide if

- you'd like to know **how the core team works**
- you'd like to know **how to reach the core team**
- you'd like to know **how to submit a bug report or feature request**
- you'd like to know **how to take part in core development by submitting a patch or pull request**
- you'd like to know **how to try and get your plugin included in Piwik Core**

### Guide assumptions

**This guide makes no assumptions.** You do not need to know how to code or know how Piwik works in order to understand this guide.

## How we manage our work

We use [trac](#) to keep track of all bugs, feature requests, and other tasks concerning Piwik, the website and all documentation.

We make sure all tickets contain enough information, including:

- if a bug, details regarding how to reproduce it,
- if a new feature, some sort of specification,
- if a UI improvement, mockups or detailed a description of the changes.

We are rather obsessed with keeping trac an organized place. Tickets are generally prioritized tickets by severity (e.g. bugs). Developers (Piwik team members or external contributors) decide for themselves which features they would like to implement.

While we are aiming to implement the features in our roadmap, we are also open to accept new [unplanned features](#) when the code and documentation meet our quality standards.

## Our release process

A release is ready when all assigned tickets [to the corresponding roadmap](#) are closed. When this happens, and the release is considered **stable**, the release is tagged with the release number (see [all tags](#)).

A shell script is run by the release manager to generate the archives (zip and tar.gz) which are copied to the build server [builds.piwik.org](#). The file [builds.piwik.org/LATEST](#) is updated with the latest stable release number. Within hours, thousands of Piwik installations will be updated by users via the one click upgrade mechanism 鈥? or manual upgrade.

Releases that contain the string 鈥?lpha 鈥?, 鈥?ta 鈥?, 鈥?c 鈥?, are built for testing purposes and are not advertised on [piwik.org](#). They are, however, made available on the build server and the [builds.piwik.org/LATEST\\_BETA](#) is updated to contain the release's version string. You can enable Piwik to use the latest Beta release automatically if you want to test the latest features ([see this faq to learn how](#)).

## Source Code Management

The Piwik git repository is hosted at [Github](#) and is publicly accessible at <https://github.com/piwik/piwik>.

In case Github goes down, we maintain a backup Git Mirror at: [git.piwik.org](#).

### Git Owners

All developers from the [Piwik team](#) can commit to the git repo.

- Matthieu Aubry ([mattab](#)) is Release manager.
- Thomas Steur ([tsteur](#)) owns the [Piwik Mobile app](#).
- Cyril B. (CyrilB) is our resident Python guru and owns the [Log Analytics](#) tool.

## Git commit process

All code committed to git is reviewed by at least one other developer in the team. Very often, Piwik developers themselves will send bigger changes by pull request for review before committing. All pull requests or patches submitted by external developers are extensively reviewed.

It is highly recommended that code committed in the [master branch](#) respects the [Piwik coding standards](#), does not cause tests to fail, and does not create regressions in the UI. It is also highly recommended that the [UI be manually tested](#) if the user interface is affected by the change. Finally, the commit message should reference a ticket number when applicable; for example,

```
fixes #159 - changed patch to use wrapInner() instead of wrap()
```

This message will automatically close the ticket [#159](#) in trac. You can also use the magic keyword

```
Refs #159
```

and a comment will be automatically added to the ticket #159 with a link to the commit on Github.

When applicable, the related [online documentation](#) and the related [FAQs](#) should be updated.

## Participating in core development

There are many ways to participate in core development, most of which do not include coding. Read on to learn all the ways you can contribute to Piwik:

### Submitting a bug report

One way to help core development is to submit a report when you find a bug.

If you believe you have found a bug in Piwik, please do the following:

- make sure you are using the latest [Piwik release](#)
- search in the [forum](#), [FAQ](#) and the [bug tracker](#) if a similar or the same bug has already been reported.
- if your bug seems new, do you understand how to reproduce it?
- if you are ready to report a bug, register an account [in the bug tracker](#), login and create a new ticket
- make sure the title and description are as descriptive and clear as possible. In the bug description, please post instructions on how to reproduce, data sets that show

the error if possible, screenshots, what exactly is not working? Is the issue new to you, or has it always failed? If you give a clear description, you will greatly help developers trying to reproduce and fix the issue.

## Submitting a feature request

Another way to contribute is to submit a feature request when you realize there is something you need that is missing in Piwik.

You can tell us what we can do to improve Piwik in the [Feature Suggestions forum](#). Please check that it is not already in the [list of Piwik tickets](#).

When submitting a significant new feature, it is highly recommended to be as descriptive as possible when creating a ticket. The ticket should contain

- a description of the product vision
- a few use cases to show how useful this feature would be
- mockups of what the new screens would be and how the existing screens would have to change
- if applicable, examples of how the feature is implemented in other existing tools

Please put as much information as possible as it will help in estimating the effort of the task and in determining how doable it would be (by the Piwik core team or a Piwik consultant). We will help with any technical details and questions outlined in the ticket.

## Contributing code

And of course if you can code and want to directly help Piwik development, you can contribute changes. To learn about contributing code changes, read our [Contributing to Piwik Core](#) guide.

## Submitting a plugin

If you've already developed a plugin that you think should be included in Piwik Core, you can offer it for inclusion. The adoption of a plugin into the Piwik core requires that we consider such criteria as (but not limited to):

- audience 鈦◆ plugin appeals to a broad spectrum of users
- desirability 鈦◆ is it a frequently requested feature by the Piwik community?
- functionality 鈦◆ feature completeness
- testability 鈦◆ use of unit tests and impact to manual testing (e.g., differences when plugin is activated vs deactivated)
- maturity 鈦◆ history and popularity of the plugin

- performance 鈥? impact on archiving and/or UI interaction
- supportability 鈥? likelihood of spawning support tickets and forum posts of the 鈥渉ow do I?鈥? or 鈥渙hy does it?鈥? variety
- complexity 鈥? simpler is better; +1 if developer has git commit privileges
- dependencies 鈥? does it depend on closed source and/or paid subscription services?
- licensing 鈥? license compatibility with GPLv3

In most cases, it should be enough for your plugin to be available on the [marketplace](#).

## Becoming a Core Contributor

If you want to become a core contributor and gain commit access to our git repository, simply continue to contribute patches and pull requests. When a certain amount are accepted and we trust your skills and judgement, we'll give you commit and add you to the core team.

## Getting in touch with the Core Team

There are several ways to talk to the Piwik team:

### Through our mailing lists

There are three mailing lists you can subscribe to and use to communicate with core team members:

- piwik-hackers [Archives](#), [Subscribe](#), [Search the archives](#)

This mailing list is used to discuss Piwik internals, ask questions about the architecture, discuss new ideas, etc.

- piwik-git [Archives](#) (or [older svn](#)), [Subscribe](#), [Search the archives](#)

This mailing list notifies all the commits to our [Git repository](#).

- piwik-trac [Archives](#), [Subscribe](#), [Search the archives](#)

This mailing list notifies all the changes to tickets in Trac (new ticket, closed, modifications, new comments, etc.)

## Using IRC

Some core team members are available via IRC at [irc.freenode.net/#piwik](#) ([webchat](#)).

# API Reference

## PHP Documentation

### Classes

*View reference docs for every Piwik class that plugin developers should use.* [Browse »](#)

### Events

*View reference docs for every event posted by Piwik and its Core Plugins.* [Browse »](#)

### Index

*View every class and method in an alphabetized index.* [Browse »](#)

### PHP Tracking client

*View reference docs for the PHP tracking client.* [Browse »](#)

## JavaScript Documentation

### Javascript Tracking client

*View reference docs for the Javascript tracking client.* [Browse »](#)

## HTTP API Documentation

### Tracking HTTP API

*View reference docs for the Tracking HTTP API.* [Browse »](#)

## Reporting API Listing

*View every API method exposed in and every parameter supported by Piwik's Reporting API.* [Browse »](#)

## Metadata

*View information about API methods that query report metadata.* [Browse »](#)

## Segmentation

*View all available segment dimensions and segment operators.* [Browse »](#)

# Classes

---

This is a complete list of available classes:

- 

### API\Request

— Dispatches API requests to the appropriate API method.

- 

### Archive

— The **Archive** class is used to query cached analytics statistics (termed "archive data").

- 

### ArchiveProcessor

— Used by [Plugin\Archiver](#) instances to insert and aggregate archive data.

- 

### ArchiveProcessor\Parameters

- Contains the analytics parameters for the reports that are currently being archived.
- 

### Common

- Contains helper methods used by both Piwik Core and the Piwik Tracking engine.
- 

### Config

- Singleton that provides read & write access to Piwik's INI configuration.
- 

### DataAccess\LogAggregator

- Contains methods that calculate metrics by aggregating log data (visits, actions, conversions, ecommerce items).
- 

### DataTable

- The primary data structure used to store analytics data in Piwik.
- 

### DataTable\BaseFilter

- A filter is set of logic that manipulates a DataTable.
- 

### DataTable\Filter\AddColumnsProcessedMetrics

- Adds processed metrics columns to a DataTable using metrics that already exist.
- 

### DataTable\Filter\AddColumnsProcessedMetricsGoal

- Adds goal related metrics to a DataTable using metrics that already exist.
- 
- DataTable\Filter\AddSummaryRow**
  - Adds a summary row to DataTables that contains the sum of all other table rows.
- 
- DataTable\Filter\BeautifyRangeLabels**
  - A DataTable filter that replaces range label columns with prettier, human-friendlier versions.
- 
- DataTable\Filter\BeautifyTimeRangeLabels**
  - A DataTable filter that replaces range labels whose values are in seconds with prettier, human-friendlier versions.
- 
- DataTable\Filter\CalculateEvolutionFilter**
  - A DataTable filter that calculates the evolution of a metric and adds it to each row as a percentage.
- 
- DataTable\Filter\ColumnCallbackAddColumn**
  - Adds a new column to every row of a DataTable based on the result of callback.
- 
- DataTable\Filter\ColumnCallbackAddColumnPercentage**
  - Calculates a percentage value for each row of a DataTable and adds the result to each row.
-

### `DataTable\Filter\ColumnCallbackAddColumnQuotient`

— Calculates the quotient of two columns and adds the result as a new column for each row of a DataTable.

- 

### `DataTable\Filter\ColumnCallbackAddMetadata`

— Executes a callback for each row of a DataTable and adds the result as a new row metadata value.

- 

### `DataTable\Filter\ColumnCallbackDeleteRow`

— Deletes all rows for which a callback returns true.

- 

### `DataTable\Filter\ColumnDelete`

— Filter that will remove columns from a DataTable using either a blacklist, whitelist or both.

- 

### `DataTable\Filter\ExcludeLowPopulation`

— Deletes all rows for which a specific column has a value that is lower than specified minimum threshold value.

- 

### `DataTable\Filter\GroupBy`

— DataTable filter that will group DataTable rows together based on the results of a reduce function.

- 

### `DataTable\Filter\Limit`

— Delete all rows from the table that are not in the given [offset, offset+limit) range.

- 

**DataTable\Filter\MetadataCallbackAddMetadata**

— Executes a callback for each row of a DataTable and adds the result to the row as a metadata value.

- 

**DataTable\Filter\MetadataCallbackReplace**

— Execute a callback for each row of a DataTable passing certain column values and metadata as metadata, and replaces row metadata with the callback result.

- 

**DataTable\Filter\Pattern**

— Deletes every row for which a specific column does not match a supplied regex pattern.

- 

**DataTable\Filter\PatternRecursive**

— Deletes rows that do not contain a column that matches a regex pattern and do not contain a subtable that contains a column that matches a regex pattern.

- 

**DataTable\Filter\ReplaceColumnNames**

— Replaces column names in each row of a table using an array that maps old column names new ones.

- 

**DataTable\Filter\ReplaceSummaryRowLabel**

— Replaces the label of the summary row with a supplied label.

- 

**DataTable\Filter\Sort**

— Sorts a DataTable based on the value of a specific column.

- 

### `DataTable\Filter\Truncate`

— Truncates a DataTable by merging all rows after a certain index into a new summary row.

- 

### `DataTable\Map`

— Stores an array of DataTables indexed by one type of DataTable metadata (such as site ID or period).

- 

### `DataTable\Row`

— This is what a [DataTable](#) is composed of.

- 

### `DataTable\Simple`

— A [DataTable](#) where every row has two columns: **label** and **value**.

- 

### `Date`

— Utility class that wraps date/time related PHP functions.

- 

### `Db`

— Contains SQL related helper functions for Piwik's MySQL database.

- 

### `Filesystem`

— Contains helper functions that deal with the filesystem.

-

## FrontController

— This singleton dispatches requests to the appropriate plugin Controller.

- 

## Http

— Contains HTTP client related helper methods that can retrieve content from remote servers and optionally save to a local file.

- 

## IP

— Contains IP address helper functions (for both IPv4 and IPv6).

- 

## Log

— Logging utility class.

- 

## Mail

— Class for sending mails, for more information

see:<http://framework.zend.com/manual/en/zend.mail.html>

- 

## Menu\MenuAbstract

— Base class for classes that manage one of Piwik's menus.

- 

## Menu\MenuAdmin

— Contains menu entries for the Admin menu.

- 

## Menu\MenuMain

- Contains menu entries for the Main menu (the menu displayed under the Piwik logo).
  -
- Menu\MenuTop**
  - Contains menu entries for the Top menu (the menu at the very top of the page).
  -
- Metrics**
  - This class contains metadata regarding core metrics and contains several related helper functions.
  -
- MetricsFormatter**
  - Contains helper function that format numerical values in different ways.
  -
- NoAccessException**
  - Exception thrown when a user doesn't have sufficient access to a resource.
  -
- Nonce**
  - Nonce class.
  -
- Notification**
  - Describes a UI notification.
  -
- Notification\Manager**
  - Posts and removes UI notifications (see [Notification](#) to learn more).

- 

## Option

— Convenient key-value storage for user specified options and temporary data that needs to be persisted beyond one request.

- 

## Period

— Date range representation.

- 

## Period\Range

— Arbitrary date range representation.

- 

## Piwik

— Main piwik helper class.

- 

## Plugin

— Base class of all Plugin Descriptor classes.

- 

## Plugin\API

— The base class of all API singletons.

- 

## Plugin\Archiver

— The base class that should be extended by plugins that compute their own analytics data.

-

## Plugin\Controller

— Base class of all plugin Controllers.

- 

## Plugin\Manager

— The singleton that manages plugin loading/unloading and installation/uninstallation.

- 

## Plugin\Settings

— Base class of all plugin settings providers.

- 

## Plugin\ViewDataTable

— The base class of all report visualizations.

- 

## Plugin\Visualization

— The base class for report visualizations that output HTML and use JavaScript.

- 

## Plugins\Actions\Archiver

— Class encapsulating logic to process Day/Period Archiving for the Actions reports

- 

## Plugins\CoreVisualizations\Visualizations\Cloud

— Generates a tag cloud from a given data array.

- 

## Plugins\CoreVisualizations\Visualizations\Graph

- This is an abstract visualization that should be the base of any 'graph' visualization.
- `Plugins\CoreVisualizations\Visualizations\HtmlTable`

  - DataTable visualization that shows DataTable data in an HTML table.
- `Plugins\CoreVisualizations\Visualizations\HtmlTable\AllColumns`

  - DataTable Visualization that derives from HtmlTable and sets `show_extra_columns` to true.
- `Plugins\CoreVisualizations\Visualizations\JqplotGraph`

  - DataTable visualization that displays DataTable data in a JQPlot graph.
- `Plugins\CoreVisualizations\Visualizations\JqplotGraph\Bar`

  - Visualization that renders HTML for a Bar graph using jqPlot.
- `Plugins\CoreVisualizations\Visualizations\JqplotGraph\Evolution`

  - Visualization that renders HTML for a line graph using jqPlot.
- `Plugins\CoreVisualizations\Visualizations\JqplotGraph\Pie`

  - Visualization that renders HTML for a Pie graph using jqPlot.
- `Plugins\CustomVariables\Archiver`

- 

```
Plugins\DevicesDetection\Archiver
```

- 

```
Plugins\ExampleVisualization\SimpleTable
```

— SimpleTable Visualization.

- 

```
Plugins\Goals\Archiver
```

- 

```
Plugins\Goals\Visualizations\Goals
```

— DataTable Visualization that derives from HtmlTable and sets show\_goals\_columns to true.

- 

```
Plugins\Live\VisitorLog
```

— A special DataTable visualization for the Live.getLastVisitsDetails API method.

- 

```
Plugins\Provider\Archiver
```

- 

```
Plugins\Referrers\Archiver
```

-

## Plugins\UserCountry\Archiver

- 

## Plugins\UserSettings\Archiver

— Archiver for UserSettings Plugin

- 

## Plugins\VisitTime\Archiver

- 

## Plugins\VisitorInterest\Archiver

- 

## RankingQuery

— The ranking query class wraps an arbitrary SQL query with more SQL that limits the number of results while aggregating the rest in an a new "Others" row.

- 

## ScheduledTask

— Contains metadata referencing PHP code that should be executed at regular intervals.

- 

## ScheduledTime

— Describes the interval on which a scheduled task is executed.

- 

## ScheduledTime\Daily

- Daily class is used to schedule tasks every day.
- 
- ScheduledTime\Hourly**
  - Hourly class is used to schedule tasks every hour.
- 
- ScheduledTime\Monthly**
  - Monthly class is used to schedule tasks every month.
- 
- ScheduledTime\Weekly**
  - Weekly class is used to schedule tasks every week.
- 
- Segment**
  - Limits the set of visits Piwik uses when aggregating analytics data.
- 
- SettingsPiwik**
  - Contains helper methods that can be used to get common Piwik settings.
- 
- SettingsServer**
  - Contains helper methods that can be used to get information regarding the server, its settings and currently used PHP settings.
- 
- Settings\Setting**
  - Base setting type class.
-

## Settings\SystemSetting

- Describes a system wide setting.
- 

## Settings\UserSetting

- Describes a per user setting.
- 

## Singleton

- The singleton base class restricts the instantiation of derived classes to one object only.
- 

## Site

- Provides access to individual [site entity](#) data (including name, URL, etc.).
- 

## TaskScheduler

- Manages scheduled task execution.
- 

## Url

- Provides URL related helper methods.
- 

## UrlHelper

- Contains less commonly needed URL helper methods.
- 

## Version

- Piwik version information.

- 

### `View`

— Encapsulates and manages a [Twig](#) template.

- 

### `ViewDataTable\Config`

— Contains base display properties for [Plugin\ViewDataTable](#)s.

- 

### `ViewDataTable\Factory`

— Provides a means of creating [Plugin\ViewDataTable](#) instances by ID.

- 

### `ViewDataTable\RequestConfig`

— Contains base request properties for [Plugin\ViewDataTable](#) instances.

- 

### `WidgetsList`

— Manages the global list of reports that can be displayed as dashboard widgets.

# Hooks

This is a complete list of available hooks.

# API

- [API.\\$pluginName.\\$methodName](#)
- [API.\\$pluginName.\\$methodName.end](#)
- [API.getReportMetadata](#)
- [API.getReportMetadata.end](#)
- [API.getSegmentDimensionMetadata](#)

- [API.Request.authenticate](#)
- [API.Request.dispatch](#)
- [API.Request.dispatch.end](#)

## API.\$pluginName.\$methodName

*Defined in [Piwik/API/Proxy](#) in line 209*

Triggered before an API request is dispatched. This event exists for convenience and is triggered directly after the [API.Request.dispatch](#) event is triggered. It can be used to modify the arguments passed to a **single** API method.

*Note: This is can be accomplished with the [API.Request.dispatch](#) event as well, however event handlers for that event will have to do more work.*

### Example

```
Piwik::addAction('API.Actions.getPageUrls', function (&$parameters) {
    // force use of a single website. for some reason.
    $parameters['idSite'] = 1;
});
```

Callback Signature:

```
function(&$finalParameters)
```

- **array**

```
&$finalParameters
```

List of parameters that will be passed to the API method.

## API.\$pluginName.\$methodName.end

*Defined in [Piwik/API/Proxy](#) in line 259*

Triggered directly after an API request is dispatched. This event exists for convenience and is triggered immediately before the [API.Request.dispatch.end](#) event. It can be used to modify the output of a **single** API method.

*Note: This can be accomplished with the [API.Request.dispatch.end](#) event as well, however event handlers for that event will have to do more work.*

## Example

```
// append (0 hits) to the end of row labels whose row has 0 hits
Piwik::addAction('API.Actions.getPageUrls', function (&$returnValue, $info)) {

    $returnValue->filter('ColumnCallbackReplace', 'label', function ($label, $hits) {

        if ($hits === 0) {

            return $label . " (0 hits)";

        } else {

            return $label;

        }

    }, null, array('nb_hits'));

}
```

Callback Signature:

`$endHookParams`

- `mixed`

`$returnedValue`

The API method's return value. Can be an object, such as a[DataTable](#) instance. Could be a [DataTable](#).

- `array`

`$extraInfo`

An array holding information regarding the API request. Will contain the following data:  
- **className**: The namespace-d class name of the API instance that's being called.  
- **module**: The name of the plugin the API request was dispatched to.  
- **action**: The name of the API method that was executed.  
- **parameters**: The array of parameters passed to the API method.

## API.getReportMetadata

Defined in [Piwik/Plugins/API/ProcessedReport](#) in line 134

Triggered when gathering metadata for all available reports. Plugins that define new reports should use this event to make them available in via the metadata API. By doing so, the report will become available in scheduled reports as well as in the Piwik Mobile App. In fact, any third party app that uses the metadata API will automatically have access to the new report.

Callback Signature:

```
function(&$availableReports, $parameters)
```

- string

```
&$availableReports
```

The list of available reports. Append to this list to make a report available. Every element of this array must contain the following information: - **category**: A translated string describing the report's category. - **name**: The translated display title of the report. - **module**: The plugin of the report. - **action**: The API method that serves the report. The following information is optional: - **dimension**: The report's [dimension](#) if any. - **metrics**: An array mapping metric names with their display names. - **metricsDocumentation**: An array mapping metric names with their translated documentation. - **processedMetrics**: The array of metrics in the report that are calculated using existing metrics. Can be set to

```
false
```

if the report contains no processed metrics. - **order**: The order of the report in the list of reports with the same category.

- array

```
$parameters
```

Contains the values of the sites and period we are getting reports for. Some reports depend on this data. For example, Goals reports depend on the site IDs being requested. Contains the following information: - **idSites**: The array of site IDs we are getting reports for. - **period**: The period type, eg,

```
'day'
```

,

'week'

,

'month'

,

'year'

,

'range'

. -**date**: A string date within the period or a date range, eg,

'2013-01-01'

or

'2012-01-01,2013-01-01'

. TODO: put dimensions section in all about analytics data

Usages:

[Actions::getReportMetadata](#), [CustomVariables::getReportMetadata](#), [DevicesDetection::getReportMetadata](#), [MultiSites::getReportMetadata](#), [Provider::getReportMetadata](#), [Referrers::getReportMetadata](#), [UserCountry::getReportMetadata](#), [UserSettings::getReportMetadata](#), [VisitFrequency::getReportMetadata](#), [VisitTime::getReportMetadata](#), [VisitorInterest::getReportMetadata](#), [VisitsSummary::getReportMetadata](#)

## API.getReportMetadata.end

Defined in [Piwik/Plugins/API/ProcessedReport](#) in line 172

Triggered after all available reports are collected. This event can be used to modify the report metadata of reports in other plugins. You could, for example, add custom metrics to every report or remove reports from the list of available reports.

Callback Signature:

`function(&$availableReports, $parameters)`

- **array**

```
&$availableReports
```

List of all report metadata. Read the [API.getReportMetadata](#)docs to see what this array contains.

- **array**

```
$parameters
```

Contains the values of the sites and period we are getting reports for. Some report depend on this data. For example, Goals reports depend on the site IDs being request. Contains the following information:

- **idSites**: The array of site IDs we are getting reports for.
- **period**: The period type, eg,

```
'day'
```

```
,
```

```
'week'
```

```
,
```

```
'month'
```

```
,
```

```
'year'
```

```
,
```

```
'range'
```

. -**date**: A string date within the period or a date range, eg,

```
'2013-01-01'
```

or

```
'2012-01-01,2013-01-01'
```

.

Usages:

[Goals::getReportMetadata](#)

## API.getSegmentDimensionMetadata

Defined in [Piwik/Plugins/API/API](#) in line [133](#)

Triggered when gathering all available segment dimensions. This event can be used to make new segment dimensions available.

### Example

```
public function getSegmentsMetadata(&$segments, $idSites)
{
    $segments[] = array(
        'type'          => 'dimension',
        'category'      => Piwik::translate('General_Visit'),
        'name'          => 'General_VisitorIP',
        'segment'       => 'visitIp',
        'acceptedValues' => '13.54.122.1, etc.',
        'sqlSegment'    => 'log_visit.location_ip',
        'sqlFilter'     => array('Piwik\IP', 'P2N'),
        'permission'    => $isAuthenticatedWithViewAccess,
    );
}
```

Callback Signature:

```
function(&$segments, $idSites)
```

- **array**

```
$dimensions
```

The list of available segment dimensions. Append to this list to add new segments.  
Each element in this list must contain the following information: - **type**: Either

```
'metric'
```

or

```
'dimension'
```

```
'metric'
```

means the value is a numeric and

```
'dimension'
```

means it is a string. Also,

```
'metric'
```

values will be displayed under **Visit (metrics)** in the Segment Editor. - **category**: The segment category name. This can be an existing segment category visible in the segment editor. - **name**: The pretty name of the segment. Can be a translation token. - **segment**: The segment name, eg,

```
'visitIp'
```

or

```
'searches'
```

. - **acceptedValues**: A string describing one or two example values, eg

```
'13.54.122.1, etc.'
```

. - **sqlSegment**: The table column this segment will segment by. For example,

```
'log_visit.location_ip'
```

for the **visitIp** segment. - **sqlFilter**: A PHP callback to apply to segment values before they are used in SQL. - **permission**: True if the current user has view access to this segment, false if otherwise.

- **array**

```
$idSites
```

The list of site IDs we're getting the available segments for. Some segments (such as Goal segments) depend on the site.

Usages:

[Actions::getSegmentsMetadata](#), [CustomVariables::getSegmentsMetadata](#), [DevicesDetection::getSegmentsMetadata](#), [Events::getSegmentsMetadata](#), [Goals::getSegmentsMetadata](#), [Provider::getSegmentsMetadata](#), [Referrers::getSegmentsMetadata](#), [UserCountry::getSegmentsMetadata](#), [UserSettings::getSegmentsMetadata](#), [VisitTime::getSegmentsMetadata](#)

## API.Request.authenticate

Defined in [Piwik/API/Request](#) in line 265

Triggered when authenticating an API request, but only if the **token\_auth** query parameter is found in the request. Plugins that provide authentication capabilities should subscribe to this event and make sure the global authentication object (the object returned by

```
Registry::get('auth')
) is setup to use
$token_auth
when its
authenticate()
method is executed.
```

Callback Signature:

```
function($token_auth)
• string
```

\$token\_auth

The value of the **token\_auth** query parameter.

Usages:

[Login::ApiRequestAuthenticate](#)

## API.Request.dispatch

Defined in [Piwik/API/Proxy](#) in line 189

Triggered before an API request is dispatched. This event can be used to modify the arguments passed to one or more API methods.

### Example

```
Piwik::addAction('API.Request.dispatch', function (&$parameters, $pluginName, $methodName) {  
    if ($pluginName == 'Actions') {  
        if ($methodName == 'getPageUrls') {  
            // ... do something ...  
        } else {  
            // ... do something else ...  
        }  
    }  
});
```

Callback Signature:

```
function(&$finalParameters, $pluginName, $methodName)
```

- `array`

```
&$finalParameters
```

List of parameters that will be passed to the API method.

- `string`

```
$pluginName
```

The name of the plugin the API method belongs to.

- `string`

```
$methodName
```

The name of the API method that will be called.

## API.Request.dispatch.end

Defined in [Piwik/API/Proxy](#) in line 299

Triggered directly after an API request is dispatched. This event can be used to modify the output of any API method.

### Example

```
// append (0 hits) to the end of row labels whose row has 0 hits
// for any report that has the 'nb_hits' metric

Piwik::addAction('API.Actions.getPageUrls', function (&$returnValue, $info) {

    // don't process non-DataTable reports and reports that don't have the nb_hits column

    if (!($returnValue instanceof DataTableInterface)
        || in_array('nb_hits', $returnValue->getColumns()))
    ) {

        return;
    }

    $returnValue->filter('ColumnCallbackReplace', 'label', function ($label, $hits) {
        if ($hits === 0) {
            return $label . " (0 hits)";
        } else {
            return $label;
        }
    }, null, array('nb_hits'));
}

}
```

Callback Signature:

`$endHookParams`

- `mixed`

```
$returnedValue
```

The API method's return value. Can be an object, such as a[DataTable](#) instance.

- [array](#)

```
$extraInfo
```

An array holding information regarding the API request. Will contain the following data:

- **className**: The namespace-d class name of the API instance that's being called.
- **module**: The name of the plugin the API request was dispatched to.
- **action**: The name of the API method that was executed.
- **parameters**: The array of parameters passed to the API method.

## ArchiveProcessor

- [ArchiveProcessor.Parameters.getIdSites](#)

### ArchiveProcessor.Parameters.getIdSites

Defined in [Piwik/ArchiveProcessor/Parameters](#) in line [111](#)

Callback Signature:

```
function(&$idSites, $this->getPeriod())
```

## AssetManager

- [AssetManager.filterMergedJavaScripts](#)
- [AssetManager.filterMergedStylesheets](#)
- [AssetManager.getJavaScriptFiles](#)
- [AssetManager.getStyleSheetFiles](#)

### AssetManager.filterMergedJavaScripts

Defined in [Piwik/AssetManager/UIAssetMerger/JScriptUIAssetMerger](#) in line [73](#)

Triggered after all the JavaScript files Piwik uses are minified and merged into a single file, but before the merged JavaScript is written to disk. Plugins can use this event to modify merged JavaScript or do something else with it.

Callback Signature:

```
function(&$mergedContent)
```

- string

```
&$mergedContent
```

The minified and merged JavaScript.

## AssetManager.filterMergedStylesheets

Defined in [Piwik/AssetManager/UIAssetMerger/StylesheetUIAssetMerger](#) in line [77](#)

Triggered after all less stylesheets are compiled to CSS, minified and merged into one file, but before the generated CSS is written to disk. This event can be used to modify merged CSS.

Callback Signature:

```
function(&$mergedContent)
```

- string

```
&$mergedContent
```

The merged and minified CSS.

## AssetManager.getJavaScriptFiles

Defined in [Piwik/AssetManager/UIAssetFetcher/JScriptUIAssetFetcher](#) in line [49](#)

Triggered when gathering the list of all JavaScript files needed by Piwik and its plugins. Plugins that have their own JavaScript should use this event to make those files load in the browser.

JavaScript files should be placed within a **javascripts** subdirectory in your plugin's root directory.

*Note: While you are developing your plugin you should enable the config setting*

```
[Debug] disable_merged_assets
```

*so JavaScript files will be reloaded immediately after every change.*

## Example

```
public function getJsFiles(&$jsFiles)
{
    $jsFiles[] = "plugins/MyPlugin/javascripts/myfile.js";
    $jsFiles[] = "plugins/MyPlugin/javascripts/anotherone.js";
}
```

Callback Signature:

```
function(&$this->fileLocations)
```

- string

```
$jsFiles
```

The JavaScript files to load.

Usages:

[Actions::getJsFiles](#), [Annotations::getJsFiles](#), [CoreAdminHome::getJsFiles](#), [CoreHome::getJsFiles](#), [CorePluginsAdmin::getJsFiles](#), [CoreVisualizations::getJsFiles](#), [Dashboard::getJsFiles](#), [ExamplePlugin::getJsFiles](#), [Feedback::getJsFiles](#), [Goals::getJsFiles](#), [LanguagesManager::getJsFiles](#), [Live::getJsFiles](#), [MobileMessaging::getJsFiles](#), [MultiSites::getJsFiles](#), [Overlay::getJsFiles](#), [PrivacyManager::getJsFiles](#), [ScheduledReports::getJsFiles](#), [SegmentEditor::getJsFiles](#), [SitesManager::getJsFiles](#), [Transitions::getJsFiles](#), [UserCountry::getJsFiles](#), [UserCountryMap::getJsFiles](#), [UsersManager::getJsFiles](#), [Widgetize::getJsFiles](#)

## AssetManager.getStylesheetFiles

Defined in [Piwik/AssetManager/UIAssetFetcher/StylesheetUIAssetFetcher](#) in line 57

Triggered when gathering the list of all stylesheets (CSS and LESS) needed by Piwik and its plugins. Plugins that have stylesheets should use this event to make those stylesheets load.

Stylesheets should be placed within a **stylesheets** subdirectory in your plugin's root directory.

*Note: While you are developing your plugin you should enable the config setting*

```
[Debug] disable_merged_assets  
so your stylesheets will be reloaded immediately after a change.
```

## Example

```
public function getStylesheetFiles(&$stylesheets)
{
    $stylesheets[] = "plugins/MyPlugin/stylesheets/myfile.less";
    $stylesheets[] = "plugins/MyPlugin/stylesheets/myotherfile.css";
}
```

Callback Signature:

```
function(&$this->fileLocations)
```

- string

```
$stylesheets
```

The list of stylesheet paths.

Usages:

[Plugin::getStylesheetFiles](#), [Actions::getStylesheetFiles](#), [Annotations::getStylesheetFiles](#), [CoreAdminHome::getStylesheetFiles](#), [CoreHome::getStylesheetFiles](#), [CorePluginsAdmin::getStylesheetFiles](#), [CoreVisualizations::getStylesheetFiles](#), [DBStats::getStylesheetFiles](#), [Dashboard::getStylesheetFiles](#), [ExampleRssWidget::getStylesheetFiles](#), [Feedback::getStylesheetFiles](#), [Goals::getStylesheetFiles](#), [Installation::getStylesheetFiles](#), [LanguagesManager::getStylesheetFiles](#), [Live::getStylesheetFiles](#), [MobileMessaging::getStylesheetFiles](#), [MultiSites::getStylesheetFiles](#), [SegmentEditor::getStylesheetFiles](#), [SitesManager::getStylesheetFiles](#), [Transitions::getStylesheetFiles](#), [UserCountry::getStylesheetFiles](#), [UserCountryMap::getStylesheetFiles](#), [UsersManager::getStylesheetFiles](#), [Widgetize::getStylesheetFiles](#)

# Config

- [Config.badConfigurationFile](#)
- [Config.NoConfigurationFile](#)

## Config.badConfigurationFile

Defined in [Piwik/FrontController](#) in line 346

Triggered if the INI config file has the incorrect format or if certain required configuration options are absent. This event can be used to start the installation process or to display a custom error message.

Callback Signature:

```
function($exception)
```

- **Exception**

```
$exception
```

The exception thrown from creating and testing the database connection.

Usages:

[Installation::dispatch](#)

## Config.NoConfigurationFile

*Defined in [Piwik/FrontController](#) in line 268*

Triggered when the configuration file cannot be found or read, which usually means Piwik is not installed yet. This event can be used to start the installation process or to display a custom error message.

Callback Signature:

```
function($exception)
```

- **Exception**

```
$exception
```

The exception that was thrown by

```
Config::getInstance()
```

Usages:

[Installation::dispatch](#)

# Console

- [Console.addCommands](#)

## Console.addCommands

Defined in [Piwik\Console](#) in line [63](#)

Triggered to gather all available console commands. Plugins that want to expose new console commands should subscribe to this event and add commands to the incoming array.

### Example

```
public function addConsoleCommands(&$commands)
{
    $commands[] = 'Piwik\Plugins\MyPlugin\Commands\MyCommand';
}
```

Callback Signature:

```
function(&$commands)
```

- **array**

```
&$commands
```

An array containing a list of command class names.

Usages:

[CoreConsole::addConsoleCommands](#), [ExampleCommand::addConsoleCommands](#), [LanguagesManager::addConsoleCommands](#)

# Controller

- [Controller.\\$module.\\$action](#)
- [Controller.\\$module.\\$action.end](#)

## Controller.\$module.\$action

Defined in [Piwik/FrontController](#) in line 114

Triggered directly before controller actions are dispatched. This event exists for convenience and is triggered directly after the [Request.dispatch](#) event is triggered.

It can be used to do the same things as the [Request.dispatch](#) event, but for one controller action only. Using this event will result in a little less code than [Request.dispatch](#).

Callback Signature:

```
function(&$parameters)
```

- array

```
&$parameters
```

The arguments passed to the controller action.

## Controller.\$module.\$action.end

Defined in [Piwik/FrontController](#) in line 132

Triggered after a controller action is successfully called. This event exists for convenience and is triggered immediately before the [Request.dispatch.end](#) event is triggered.

It can be used to do the same things as the [Request.dispatch.end](#) event, but for one controller action only. Using this event will result in a little less code than [Request.dispatch.end](#).

Callback Signature:

```
function(&$result, $parameters)
```

- mixed

```
&$result
```

The result of the controller action.

- array

```
$parameters
```

The arguments passed to the controller action.

## CronArchive

- [CronArchive.filterWebsitelds](#)

### CronArchive.filterWebsitelds

Defined in [Piwik/CronArchive](#) in line [806](#)

Triggered by the **archive.php** cron script so plugins can modify the list of websites that the archiving process will be launched for. Plugins can use this hook to add websites to archive, remove websites to archive, or change the order in which websites will be archived.

Callback Signature:

```
function(&$websiteIds)
```

- **array**

```
&$websiteIds
```

The list of website IDs to launch the archiving process for.

## Goals

- [Goals.getReportsWithGoalMetrics](#)

### Goals.getReportsWithGoalMetrics

Defined in [Piwik/Plugins/Goals/Goals](#) in line [393](#)

Triggered when gathering all reports that contain Goal metrics. The list of reports will be displayed on the left column of the bottom of every *Goals* page.

If plugins define reports that contain goal metrics (such as **conversions** or **revenue**), they can use this event to make sure their reports can be viewed on Goals pages.

#### Example

```
public function getReportsWithGoalMetrics(&$reports)
```

```

{
    $reports[ ] = array(
        'category' => Piwik::translate('MyPlugin_myReportCategory'),
        'name' => Piwik::translate('MyPlugin_myReportDimension'),
        'module' => 'MyPlugin',
        'action' => 'getMyReport'
    );
}

```

Callback Signature:

```
function(&$reportsWithGoals)
```

- **array**

```
&$reportsWithGoals
```

The list of arrays describing reports that have Goal metrics. Each element of this array must be an array with the following properties:

- **category**: The report category. This should be a translated string.
- **name**: The report's translated name.
- **module**: The plugin the report is in, eg,

```
'UserCountry'
```

- . - **action**: The API method of the report, eg,

```
'getCountry'
```

Usages:

[CustomVariables::getReportsWithGoalMetrics](#), [Goals::getActualReportsWithGoalMetrics](#),  
[Referrers::getReportsWithGoalMetrics](#), [UserCountry::getReportsWithGoalMetrics](#),[VisitTime::getReportsWithGoalMetrics](#)

# Live

- [Live.API.getIdSitesString](#)
- [Live.getExtraVisitorDetails](#)

## Live.API.getIdSitesString

Defined in [Piwik/Plugins/Live/API](#) in line [696](#)

Callback Signature:

```
function(&$idSites)
```

## Live.getExtraVisitorDetails

Defined in [Piwik/Plugins/Live/API](#) in line [385](#)

Triggered in the Live.getVisitorProfile API method. Plugins can use this event to discover and add extra data to visitor profiles.

For example, if an email address is found in a custom variable, a plugin could load the gravatar for the email and add it to the visitor profile, causing it to display in the visitor profile popup.

The following visitor profile elements can be set to augment the visitor profile popup:

- **visitorAvatar**: A URL to an image to display in the top left corner of the popup.
- **visitorDescription**: Text to be used as the tooltip of the avatar image.

Callback Signature:

```
function(&$result)
```

- **array**

```
$visitorProfile
```

The unaugmented visitor profile info.

# Log

- [Log.formatDatabaseMessage](#)

- [Log.formatFileMessage](#)
- [Log.formatScreenMessage](#)
- [Log.getAvailableWriters](#)

## Log.formatDatabaseMessage

Defined in [Piwik\Log](#) in line [486](#)

Triggered when trying to log an object to a database table. Plugins can use this event to convert objects to strings before they are logged.

### Example

```
public function formatDatabaseMessage(&$message, $level, $tag,
$datetime, $logger) {
    if ($message instanceof MyCustomDebugInfo) {
        $message = $message->formatForDatabase();
    }
}
```

Callback Signature:

<code>function(&amp;\$message, \$level, \$tag, \$datetime, \$logger)</code>
-----------------------------------------------------------------------------

- `mixed`

<code>&amp;\$message</code>
-----------------------------

The object that is being logged. Event handlers should check if the object is of a certain type and if it is, set

<code>\$message</code>
------------------------

to the string that should be logged.

- `int`

<code>\$level</code>
----------------------

The log level used with this log entry.

- `string`

`$tag`

The current plugin that started logging (or if no plugin, the current class).

- `string`

`$datetime`

Datetime of the logging call.

- [Log](#)

`$logger`

The Log singleton.

## Log.formatFileMessage

Defined in [Piwik/Log](#) in line [388](#)

Triggered when trying to log an object to a file. Plugins can use this event to convert objects to strings before they are logged.

### Example

```
public function formatFileMessage(&$message, $level, $tag, $datetime, $logger) {  
    if ($message instanceof MyCustomDebugInfo) {  
        $message = $message->formatForFile();  
    }  
}
```

Callback Signature:

```
function(&$message, $level, $tag, $datetime, $logger)
```

- `mixed`

`&$message`

The object that is being logged. Event handlers should check if the object is of a certain type and if it is, set

`$message`

to the string that should be logged.

- `int`

`$level`

The log level used with this log entry.

- `string`

`$tag`

The current plugin that started logging (or if no plugin, the current class).

- `string`

`$datetime`

Datetime of the logging call.

- [Log](#)

`$logger`

The Log singleton.

## Log.formatScreenMessage

Defined in [Piwik\Log](#) in line 448

Triggered when trying to log an object to the screen. Plugins can use this event to convert objects to strings before they are logged.

The result of this callback can be HTML so no sanitization is done on the result. This means **YOU MUST SANITIZE THE MESSAGE YOURSELF** if you use this event.

### Example

```
public function formatScreenMessage(&$message, $level, $tag, $datetime, $logger) {  
    if ($message instanceof MyCustomDebugInfo) {  
        $message = Common::sanitizeInputValue($message->formatForScreen());  
    }  
}
```

Callback Signature:

```
function(&$message, $level, $tag, $datetime, $logger)
```

- mixed

```
&$message
```

The object that is being logged. Event handlers should check if the object is of a certain type and if it is, set

```
$message
```

to the string that should be logged.

- int

```
$level
```

The log level used with this log entry.

- string

**\$tag**

The current plugin that started logging (or if no plugin, the current class).

- **string**

**\$datetime**

Datetime of the logging call.

- [Log](#)

**\$logger**

The Log singleton.

## Log.getAvailableWriters

Defined in [Piwik/Log](#) in line [352](#)

This event is called when the Log instance is created. Plugins can use this event to make new logging writers available.

A logging writer is a callback with the following signature:

```
function (int $level, string $tag, string $datetime, string $message)
```

**\$level**

is the log level to use,

**\$tag**

is the log tag used,

**\$datetime**

is the date time of the logging call and

**\$message**

is the formatted log message.

Logging writers must be associated by name in the array passed to event handlers. The name specified can be used in Piwik's INI configuration.

### Example

```
public function getAvailableWriters(&$writers) {  
    $writers['myloggername'] = function ($level, $tag, $datetime, $message) {  
        // ...  
    };  
}  
  
// 'myLoggername' can now be used in the log_writers config option.
```

Callback Signature:

```
function(&$writers)
```

- array

```
&$writers
```

Array mapping writer names with logging writers.

## Menu

- [Menu.Admin.addItems](#)
- [Menu.Reporting.addItems](#)
- [Menu.Top.addItems](#)

### Menu.Admin.addItems

Defined in [Piwik\Menu\MenuAdmin](#) in line 84

Triggered when collecting all available admin menu items. Subscribe to this event if you want to add one or more items to the Piwik admin menu.

Menu items should be added via the [add\(\)](#) method.

#### Example

```
use Piwik\Menu\MenuAdmin;  
  
public function addMenuItem()
```

```

{
    MenuAdmin::getInstance()->add(
        'MenuName',
        'SubmenuName',
        array('module' => 'MyPlugin', 'action' => 'index'),
        $showOnlyIf = Piwik::isUserIsSuperUser(),
        $order = 6
    );
}

```

Usages:

[CoreAdminHome::addMenu](#), [CorePluginsAdmin::addMenu](#), [DBStats::addMenu](#), [Installations::addMenu](#), [MobileMessaging::addMenu](#), [PrivacyManager::addMenu](#), [SitesManager::addMenu](#), [UserCountry::addAdminMenu](#), [UsersManager::addMenu](#)

## Menu.Reporting.addItem

Defined in [Piwik/Menu/MenuMain](#) in line [90](#)

Triggered when collecting all available reporting menu items. Subscribe to this event if you want to add one or more items to the Piwik reporting menu.

Menu items should be added via the [add\(\)](#) method.

### Example

```

use Piwik\Menu\Main;

public function addMenuItem()
{
    Main::getInstance()->add(
        'CustomMenuName',
        'CustomSubMenuName',
        array('module' => 'MyPlugin', 'action' => 'index'),
        $showOnlyIf = Piwik::isUserIsSuperUser(),
        $order = 6
}

```

```
 );  
}
```

Usages:

[Actions::addMenus](#), [CustomVariables::addMenus](#), [Dashboard::addMenus](#), [DevicesDetection::addMenu](#), [ExampleUI::addReportingMenuItems](#), [Goals::addMenus](#), [Live::addMenu](#), [Provider::addMenu](#), [Referrers::addMenus](#), [UserCountry::addMenu](#), [UserCountryMap::addMenu](#), [UserSettings::addMenu](#), [VisitFrequency::addMenu](#), [VisitTime::addMenu](#), [VisitorInterest::addMenu](#), [VisitsSummary::addMenu](#)

## Menu.Top.addItem

Defined in [Piwik\Menu\MenuTop](#) in line [115](#)

Triggered when collecting all available menu items that are be displayed on the very top of every page, next to the login/logout links. Subscribe to this event if you want to add one or more items to the top menu.

Menu items should be added via the [addEntry\(\)](#) method.

### Example

```
use Piwik\Menu\MenuTop;  
  
public function addMenuItems()  
{  
    MenuTop::addEntry(  
        'TopMenuItemName',  
        array('module' => 'MyPlugin', 'action' => 'index'),  
        $showOnlyIf = Piwik::isUserIsSuperUser(),  
        $order = 6  
    );  
}
```

Usages:

[Plugin::addTopMenu](#), [Dashboard::addTopMenu](#), [ExampleUI::addTopMenuItems](#), [Feedback::addTopMenu](#), [LanguagesManager::showLanguagesSelector](#), [MultiSites::addTopMenu](#), [ScheduledReports::addTopMenu](#), [Widgetize::addTopMenu](#)

# Provider

- [Provider.getCleanHostname](#)

## Provider.getCleanHostname

Defined in [Piwik/Plugins/Provider/Provider](#) in line [187](#)

Triggered when prettifying a hostname string. This event can be used to customize the way a hostname is displayed in the Providers report.

### Example

```
public function getCleanHostname(&$cleanHostname, $hostname)
{
    if ('fvae.VARG.ceaga.site.co.jp' == $hostname) {
        $cleanHostname = 'site.co.jp';
    }
}
```

Callback Signature:

```
function(&$cleanHostname, $hostname)
```

- [string](#)

```
&$cleanHostname
```

The hostname string to display. Set by the event handler.

- [string](#)

```
$hostname
```

The full hostname.

# Reporting

- [Reporting.getDatabaseConfig](#)

## Reporting.getDatabaseConfig

Defined in [Piwik/Db](#) in line [92](#)

Triggered before a database connection is established. This event can be used to change the settings used to establish a connection.

Callback Signature:

```
function(&$dbInfos)
```

- [array](#)

# Request

- [Request.dispatch](#)
- [Request.dispatch.end](#)
- [Request.dispatchCoreAndPluginUpdatesScreen](#)
- [Request.initAuthenticationObject](#)
- [Request.initAuthenticationObject](#)

## Request.dispatch

Defined in [Piwik/FrontController](#) in line [99](#)

Triggered directly before controller actions are dispatched. This event can be used to modify the parameters passed to one or more controller actions and can be used to change the controller action being dispatched to.

Callback Signature:

```
function(&$module, &$action, &$parameters)
```

- [string](#)

```
&$module
```

The name of the plugin being dispatched to.

- `string`

`&$action`

The name of the controller method being dispatched to.

- `array`

`&$parameters`

The arguments passed to the controller action.

## Request.dispatch.end

Defined in [Piwik/FrontController](#) in line [142](#)

Triggered after a controller action is successfully called. This event can be used to modify controller action output (if any) before the output is returned.

Callback Signature:

`function(&$result, $parameters)`

- `mixed`

`&$result`

The controller action result.

- `array`

`$parameters`

The arguments passed to the controller action.

## Request.dispatchCoreAndPluginUpdatesScreen

Defined in [Piwik/FrontController](#) in line [360](#)

Triggered just after the platform is initialized and plugins are loaded. This event can be used to do early initialization.

*Note: At this point the user is not authenticated yet.*

Usages:

[CoreUpdater::dispatch](#)

## Request.initAuthenticationObject

Defined in [Piwik/Plugins/Overlay/API](#) in line 128

Triggered immediately before the user is authenticated. This event can be used by plugins that provide their own authentication mechanism to make that mechanism available.

Subscribers should set the

'auth'

object in the Piwik\Registry to an object that implements the Piwik\Auth interface.

### Example

```
use Piwik\Registry;

public function initAuthenticationObject($allowCookieAuthentica
tion)
{
    Registry::set('auth', new LDAPAuth($allowCookieAuthenticati
on));
}
```

Callback Signature:

`function($allowCookieAuthentication = true)`

- bool

`$allowCookieAuthentication`

Whether authentication based on

`$_COOKIE`

values should be allowed.

Usages:

[Login::initAuthenticationObject](#)

## Request.initAuthenticationObject

Defined in [Piwik\FrontController](#) in line 382

Triggered before the user is authenticated, when the global authentication object should be created. Plugins that provide their own authentication implementation should use this event to set the global authentication object (which must derive from Piwik\Auth).

### Example

```
Piwik::addAction('Request.initAuthenticationObject', function()
{
    Piwik\Registry::set('auth', new MyAuthImplementation());
});
```

Usages:

[Login::initAuthenticationObject](#)

## ScheduledReports

- [ScheduledReports.allowMultipleReports](#)
- [ScheduledReports.getRendererInstance](#)
- [ScheduledReports.getReportFormats](#)
- [ScheduledReports.getReportMetadata](#)
- [ScheduledReports.getReportParameters](#)
- [ScheduledReports.getReportRecipients](#)
- [ScheduledReports.getReportTypes](#)
- [ScheduledReports.processReports](#)
- [ScheduledReports.sendReport](#)
- [ScheduledReports.validateReportParameters](#)

## ScheduledReports.allowMultipleReports

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 776

Triggered when we're determining if a scheduled report transport medium can handle sending multiple Piwik reports in one scheduled report or not. Plugins that provide their

own transport mediums should use this event to specify whether their backend can send more than one Piwik report at a time.

Callback Signature:

```
function(&$allowMultipleReports, $reportType)
```

- `bool`

```
&$allowMultipleReports
```

Whether the backend type can handle multiple Piwik reports or not.

- `string`

```
$reportType
```

A string ID describing how the report is sent, eg,

```
'sms'
```

or

```
'email'
```

.

Usages:

[MobileMessaging::allowMultipleReports](#), [ScheduledReports::allowMultipleReports](#)

## ScheduledReports.getRendererInstance

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 425

Triggered when obtaining a renderer instance based on the scheduled report output format. Plugins that provide new scheduled report output formats should use this event to handle their new report formats.

Callback Signature:

```
function(&$reportRenderer, $reportType, $outputType, $report)
```

- `ReportRenderer`

```
&$reportRenderer
```

This variable should be set to an instance that extends `Piwik\ReportRenderer` by one of the event subscribers.

- `string`

```
$reportType
```

A string ID describing how the report is sent, eg,

```
'sms'
```

or

```
'email'
```

- `string`

```
$outputType
```

The output format of the report, eg,

```
'html'
```

,

```
'pdf'
```

, etc.

- `array`

```
&$report
```

An array describing the scheduled report that is being generated.

Usages:

[MobileMessaging::getRendererInstance](#), [ScheduledReports::getRendererInstance](#)

## ScheduledReports.getReportFormats

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 823

Triggered when gathering all available scheduled report formats. Plugins that provide their own scheduled report format should use this event to make their format available.

Callback Signature:

```
function(&$reportFormats, $reportType)
```

- `array`

```
&$reportFormats
```

An array mapping string IDs for each available scheduled report format with icon paths for those formats. Add your new format's ID to this array.

- `string`

```
$reportType
```

A string ID describing how the report is sent, eg,

```
'sms'
```

or

```
'email'
```

.

Usages:

[MobileMessaging::getReportFormats](#), [ScheduledReports::getReportFormats](#)

# ScheduledReports.getReportMetadata

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 748

TODO: change this event so it returns a list of API methods instead of report metadata arrays. Triggered when gathering the list of Piwik reports that can be used with a certain transport medium.

Plugins that provide their own transport mediums should use this event to list the Piwik reports that their backend supports.

Callback Signature:

```
function(&$availableReportMetadata, $reportType, $idSite)
```

- `array`

```
&$availableReportMetadata
```

An array containing report metadata for each supported report.

- `string`

```
$reportType
```

A string ID describing how the report is sent, eg,

```
'sms'
```

or

```
'email'
```

- `int`

```
$idSite
```

The ID of the site we're getting available reports for.

Usages:

[MobileMessaging::getReportMetadata](#), [ScheduledReports::getReportMetadata](#)

## ScheduledReports.getReportParameters

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 602

Triggered when gathering the available parameters for a scheduled report type. Plugins that provide their own scheduled report transport mediums should use this event to list the available report parameters for their transport medium.

Callback Signature:

`function(&$availableParameters, $reportType)`

- `array`

`&$availableParameters`

The list of available parameters for this report type. This is an array that maps parameter IDs with a boolean that indicates whether the parameter is mandatory or not.

- `string`

`$reportType`

A string ID describing how the report is sent, eg,

`'sms'`

or

`'email'`

.

Usages:

[MobileMessaging::getReportParameters](#), [ScheduledReports::getReportParameters](#)

# ScheduledReports.getReportRecipients

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 854

Triggered when getting the list of recipients of a scheduled report. Plugins that provide their own scheduled report transport medium should use this event to extract the list of recipients their backend's specific scheduled report format.

Callback Signature:

```
function(&$recipients, $report['type'], $report)
```

- **array**

```
&$recipients
```

An array of strings describing each of the scheduled reports recipients. Can be, for example, a list of email addresses or phone numbers or whatever else your plugin uses.

- **string**

```
$reportType
```

A string ID describing how the report is sent, eg,

```
'sms'
```

or

```
'email'
```

- **array**

```
$report
```

An array describing the scheduled report that is being generated.

Usages:

[MobileMessaging::getReportRecipients](#), [ScheduledReports::getReportRecipients](#)

## ScheduledReports.getReportTypes

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line [799](#)

Triggered when gathering all available transport mediums. Plugins that provide their own transport mediums should use this event to make their medium available.

Callback Signature:

```
function(&$reportTypes)
```

- array

```
&$reportTypes
```

An array mapping transport medium IDs with the paths to those mediums' icons.  
Add your new backend's ID to this array.

Usages:

[MobileMessaging::getReportTypes](#), [ScheduledReports::getReportTypes](#)

## ScheduledReports.processReports

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line [403](#)

Triggered when generating the content of scheduled reports. This event can be used to modify the report data or report metadata of one or more reports in a scheduled report, before the scheduled report is rendered and delivered.

TODO: list data available in \$report or make it a new class that can be documented (same for all other events that use a \$report)

Callback Signature:

```
function(&$processedReports, $reportType, $outputType, $report)
```

- array

### `&$processedReports`

The list of processed reports in the scheduled report. Entries includes report data and metadata for each report.

- `string`

### `$reportType`

A string ID describing how the scheduled report will be sent, eg,

`'sms'`

or

`'email'`

- `string`

### `$outputType`

The output format of the report, eg,

`'html'`

,

`'pdf'`

, etc.

- `array`

### `$report`

An array describing the scheduled report that is being generated.

Usages:

[ScheduledReports::processReports](#)

## ScheduledReports.sendReport

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 544

Triggered when sending scheduled reports. Plugins that provide new scheduled report transport mediums should use this event to send the scheduled report.

Callback Signature:

```
function($report['type'], $report, $contents, $filename, $prettyDate, $reportSubject, $reportTitle, $additionalFiles)
```

- `string`

`$reportType`

A string ID describing how the report is sent, eg,

`'sms'`

or

`'email'`

- `array`

`$report`

An array describing the scheduled report that is being generated.

- `string`

`$contents`

The contents of the scheduled report that was generated and now should be sent.

- `string`

`$filename`

The path to the file where the scheduled report has been saved.

- `string`

`$prettyDate`

A prettified date string for the data within the scheduled report.

- `string`

`$reportSubject`

A string describing what's in the scheduled report.

- `string`

`$reportTitle`

The scheduled report's given title (given by a Piwik user).

- `array`

`$additionalFiles`

The list of additional files that should be sent with this report.

Usages:

[MobileMessaging::sendReport](#), [ScheduledReports::sendReport](#)

## ScheduledReports.validateReportParameters

Defined in [Piwik/Plugins/ScheduledReports/API](#) in line 629

Triggered when validating the parameters for a scheduled report. Plugins that provide their own scheduled reports backend should use this event to validate the custom parameters defined with ScheduledReports::getReportParameters().

Callback Signature:

```
function(&$parameters, $reportType)
```

- `array`

```
&$parameters
```

The list of parameters for the scheduled report.

- `string`

```
$reportType
```

A string ID describing how the report is sent, eg,

```
'sms'
```

or

```
'email'
```

.

Usages:

[MobileMessaging::validateReportParameters](#), [ScheduledReports::validateReportParameters](#)

## SegmentEditor

- [SegmentEditor.deactivate](#)

### SegmentEditor.deactivate

Defined in [Piwik/Plugins/SegmentEditor/API](#) in line 309

Triggered before a segment is deleted or made invisible. This event can be used by plugins to throw an exception or do something else.

Callback Signature:

```
function($idSegment)
```

- int

```
$idSegment
```

The ID of the segment being deleted.

Usages:

[ScheduledReports::segmentDeactivation](#)

## Segments

- [Segments.getKnownSegmentsToArchiveAllSites](#)
- [Segments.getKnownSegmentsToArchiveForSite](#)

### Segments.getKnownSegmentsToArchiveAllSites

Defined in [Piwik/SettingsPiwik](#) in line 91

Triggered during the cron archiving process to collect segments that should be pre-processed for all websites. The archiving process will be launched for each of these segments when archiving data.

This event can be used to add segments to be pre-processed. If your plugin depends on data from a specific segment, this event could be used to provide enhanced performance.

*Note: If you just want to add a segment that is managed by the user, use the SegmentEditor API.*

#### Example

```
Piwik::addAction('Segments.getKnownSegmentsToArchiveAllSites',  
function (&$segments) {  
    $segments[] = 'country=jp;city=Tokyo';
```

```
});
```

Callback Signature:

```
function(&$segmentsToProcess)
```

- array

```
&$segmentsToProcess
```

List of segment definitions, eg, array( 'browserCode=ff;resolution=800x600', 'country=jp;city=Tokyo' ) Add segments to this array in your event handler.

Usages:

[SegmentEditor::getKnownSegmentsToArchiveAllSites](#)

## Segments.getKnownSegmentsToArchiveForSite

Defined in [Piwik/SettingsPiwik](#) in line [136](#)

Triggered during the cron archiving process to collect segments that should be pre-processed for one specific site. The archiving process will be launched for each of these segments when archiving data for that one site.

This event can be used to add segments to be pre-processed for one site.

*Note: If you just want to add a segment that is managed by the user, you should use the SegmentEditor API.*

### Example

```
Piwik::addAction('Segments.getKnownSegmentsToArchiveForSite', function (&$segments, $idSite) {
    $segments[] = 'country=jp;city=Tokyo';
});
```

Callback Signature:

```
function(&$segments, $idSite)
```

- array

## \$segmentsToProcess

List of segment definitions, eg, array( 'browserCode=ff;resolution=800x600', 'country=JP;city=Tokyo' ) Add segments to this array in your event handler.

- int

## \$idSite

The ID of the site to get segments for.

Usages:

[SegmentEditor::getKnownSegmentsToArchiveForSite](#)

# Site

- [Site.setSite](#)

## Site.setSite

Defined in [Piwik/Site](#) in line [120](#)

Triggered so plugins can modify website entities without modifying the database. This event should **not** be used to add data that is expensive to compute. If you need to make HTTP requests or query the database for more information, this is not the place to do it.

### Example

```
Piwik::addAction('Site.setSite', function ($idSite, &$info) {  
    $info['name'] .= " (original)";  
});
```

Callback Signature:

```
function($idSite, &$infoSite)
```

- int

## \$idSite

The ID of the website entity that will be modified.

- `array`

`&$infoSite`

The website entity. [Learn more.](#)

## SitesManager

- [SitesManager.deleteSite.end](#)

### SitesManager.deleteSite.end

Defined in [Piwik/Plugins/SitesManager/API](#) in line [619](#)

Triggered after a site has been deleted. Plugins can use this event to remove site specific values or settings, such as removing all goals that belong to a specific website. If you store any data related to a website you should clean up that information here.

Callback Signature:

`function($idSite)`

- `int`

`$idSite`

The ID of the site being deleted.

Usages:

[Goals::deleteSiteGoals](#), [ScheduledReports::deleteSiteReport](#), [UsersManager::deleteSite](#)

## TaskScheduler

- [TaskScheduler.getScheduledTasks](#)

### TaskScheduler.getScheduledTasks

Defined in [Piwik/TaskScheduler](#) in line [102](#)

Triggered during scheduled task execution. Collects all the tasks to run.

Subscribe to this event to schedule code execution on an hourly, daily, weekly or monthly basis.

### Example

```
public function getScheduledTasks(&$tasks)
{
    $tasks[] = new ScheduledTask(
        'Piwik\Plugins\CorePluginsAdmin\MarketplaceApiClient',
        'clearAllCacheEntries',
        null,
        ScheduledTime::factory('daily'),
        ScheduledTask::LOWEST_PRIORITY
    );
}
```

Callback Signature:

```
function(&$tasks)
```

- [ScheduledTask](#)

```
&$tasks
```

List of tasks to run periodically.

Usages:

[CoreAdminHome::getScheduledTasks](#), [CorePluginsAdmin::getScheduledTasks](#), [DBStats::getScheduledTasks](#), [PrivacyManager::getScheduledTasks](#), [ScheduledReports::getScheduledTasks](#), [UserCountry::getScheduledTasks](#)

## Tracker

- [Tracker.Cache.getSiteAttributes](#)
- [Tracker.detectReferrerSearchEngine](#)

- [Tracker.existingVisitInformation](#)
- [Tracker.getDatabaseConfig](#)
- [Tracker.getVisitFieldsToPersist](#)
- [Tracker.isExcludedVisit](#)
- [Tracker.makeNewVisitObject](#)
- [Tracker.newConversionInformation](#)
- [Tracker.newVisitorInformation](#)
- [Tracker.recordAction](#)
- [Tracker.recordEcommerceGoal](#)
- [Tracker.recordStandardGoals](#)
- [Tracker.Request.getIdSite](#)
- [Tracker.setTrackerCacheGeneral](#)
- [Tracker.setVisitorIp](#)

## Tracker.Cache.getSiteAttributes

Defined in [Piwik/Tracker/Cache](#) in line [90](#)

Triggered to get the attributes of a site entity that might be used by the Tracker. Plugins add new site attributes for use in other tracking events must use this event to put those attributes in the Tracker Cache.

### Example

```
public function getSiteAttributes($content, $idSite)
{
    $sql = "SELECT info FROM " . Common::prefixTable('myplugin_extra_site_info') . " WHERE idsite = ?";
    $content['myplugin_site_data'] = Db::fetchOne($sql, array($idSite));
}
```

Callback Signature:

<b>function(&amp;\$content, \$idSite)</b>
• <b>array</b>

<b>&amp;\$content</b>
-----------------------

Array mapping of site attribute names with values.

- `int`

```
$idSite
```

The site ID to get attributes for.

Usages:

[Goals::fetchGoalsFromDb](#), [SitesManager::recordWebsiteDataInCache](#), [UsersManager::recordAdminUsersInCache](#)

## Tracker.detectReferrerSearchEngine

Defined in [Piwik/Tracker/Referrer](#) in line [143](#)

Triggered when detecting the search engine of a referrer URL. Plugins can use this event to provide custom search engine detection logic.

Callback Signature:

```
function(&$searchEngineInformation, $this->referrerUrl)
```

- `array`

```
&$searchEngineInformation
```

An array with the following information:  
- **name**: The search engine name.  
- **keywords**: The search keywords used. This parameter is initialized to the results of Piwik's default search engine detection logic.

- `string`

## Tracker.existingVisitInformation

Defined in [Piwik/Tracker/Visit](#) in line [256](#)

Triggered before a [visit entity](#) is updated when tracking an action for an existing visit. This event can be used to modify the visit properties that will be updated before the changes are persisted.

Callback Signature:

```
function(&$valuesToUpdate, $this->visitorInfo)
```

- **array**

```
&$valuesToUpdate
```

Visit entity properties that will be updated.

- **array**

```
$visit
```

The entire visit entity. Read [this](#) to see what it contains.

## Tracker.getDatabaseConfig

Defined in [Piwik/Tracker](#) in line [542](#)

Triggered before a connection to the database is established by the Tracker. This event can be used to change the database connection settings used by the Tracker.

Callback Signature:

```
function(&$configDb)
```

- **array**

```
$dbInfos
```

Reference to an array containing database connection info, including:  
- **host**: The host name or IP address to the MySQL database.  
- **username**: The username to use when connecting to the database.  
- **password**: The password to use when connecting to the database.  
- **dbname**: The name of the Piwik MySQL database.  
- **port**: The MySQL database port to use.  
- **adapter**: either

```
'PDO_MYSQL'
```

or

```
'MYSQLI'
```

## Tracker.getVisitFieldsToPersist

Defined in [Piwik/Tracker/Visit](#) in line [999](#)

Triggered when checking if the current action being tracked belongs to an existing visit. This event collects a list of [visit entity](#) properties that should be loaded when reading the existing visit. Properties that appear in this list will be available in other tracking events such as [Tracker.newConversionInformation](#) and [Tracker.newVisitorInformation](#).

Plugins can use this event to load additional visit entity properties for later use during tracking. When you add fields to this \$fields array, they will be later available in [Tracker.newConversionInformation](#)

### Example

```
Piwik::addAction('Tracker.getVisitFieldsToPersist', function (&
$fields) {
    $fields[] = 'custom_visit_property';
});
```

Callback Signature:

```
function(&$fields)
    • array
```

```
&$fields
```

The list of visit properties to load.

## Tracker.isExcludedVisit

Defined in [Piwik/Tracker/VisitExcluded](#) in line [84](#)

Triggered on every tracking request. This event can be used to tell the Tracker not to record this particular action or visit.

Callback Signature:

```
function(&$excluded)
    • bool
```

```
&$excluded
```

Whether the request should be excluded or not. Initialized to

```
false
```

. Event subscribers should set it to

```
true
```

in order to exclude the request.

Usages:

[DoNotTrack::checkHeader](#)

## Tracker.makeNewVisitObject

Defined in [Piwik\Tracker](#) in line [599](#)

Triggered before a new **visit tracking object** is created. Subscribers to this event can force the use of a custom visit tracking object that extends from `Piwik\Tracker\VisitInterface`.

Callback Signature:

```
function(&$visit)
```

- `\Piwik\Tracker\VisitInterface`

```
&$visit
```

Initialized to null, but can be set to a new visit object. If it isn't modified Piwik uses the default class.

## Tracker.newConversionInformation

Defined in [Piwik\Tracker\GoalManager](#) in line [815](#)

Triggered before persisting a new [conversion entity](#). This event can be used to modify conversion information or to add new information to be persisted.

Callback Signature:

```
function(&$conversion, $visitInformation, $this->request)
```

- **array**

```
&$conversion
```

The conversion entity. Read [this](#) to see what it contains.

- **array**

```
$visitInformation
```

The visit entity that we are tracking a conversion for. See what information it contains [here](#).

- **\Piwik\Tracker\Request**

```
$request
```

An object describing the tracking request being processed.

## Tracker.newVisitorInformation

Defined in [Piwik/Tracker/Visit](#) in line [310](#)

Triggered before a new [visit entity](#) is persisted. This event can be used to modify the visit entity or add new information to it before it is persisted. The UserCountry plugin, for example, uses this event to add location information for each visit.

Callback Signature:

```
function(&$this->visitorInfo, $this->request)
```

- **array**

```
$visit
```

The visit entity. Read [this](#) to see what information it contains.

- \Piwik\Tracker\Request

`$request`

An object describing the tracking request being processed.

Usages:

[DevicesDetection::parseMobileVisitData](#), [Provider::enrichVisitWithProviderInfo](#), [UserCoun](#)  
[try::enrichVisitWithLocation](#)

## Tracker.recordAction

Defined in [Piwik/Tracker/Action](#) in line [315](#)

Triggered after successfully persisting a [visit action entity](#).

Callback Signature:

```
function($trackerAction = $this, $visitAction)
```

- Action

`$tracker`

Action The Action tracker instance.

- array

`$visitAction`

The visit action entity that was persisted. Read [this](#) to see what it contains.

## Tracker.recordEcommerceGoal

Defined in [Piwik/Tracker/GoalManager](#) in line [421](#)

Triggered after successfully persisting an ecommerce conversion. Note: Subscribers should be wary of doing any expensive computation here as it may slow the tracker down.

Callback Signature:

```
function($conversion, $visitInformation)
```

- **array**

```
$conversion
```

The conversion entity that was just persisted. See what information it contains [here](#).

- **array**

```
$visitInformation
```

The visit entity that we are tracking a conversion for. See what information it contains [here](#).

## Tracker.recordStandardGoals

Defined in [Piwik/Tracker/GoalManager](#) in line [791](#)

Triggered after successfully recording a non-ecommerce conversion. Note: Subscribers should be wary of doing any expensive computation here as it may slow the tracker down.

Callback Signature:

```
function($conversion)
```

- **array**

```
$conversion
```

The conversion entity that was just persisted. See what information it contains [here](#).

## Tracker.Request.getIdSite

Defined in [Piwik/Tracker/Request](#) in line [318](#)

Triggered when obtaining the ID of the site we are tracking a visit for. This event can be used to change the site ID so data is tracked for a different website.

Callback Signature:

```
function(&$idSite, $this->params)
```

- int

```
&$idSite
```

Initialized to the value of the **idsite** query parameter. If a subscriber sets this variable, the value it uses must be greater than 0.

- array

```
$params
```

The entire array of request parameters in the current tracking request.

## Tracker.setTrackerCacheGeneral

Defined in [Piwik/Tracker/Cache](#) in line [154](#)

Triggered before the [general tracker cache](#) is saved to disk. This event can be used to add extra content to the cache.

Data that is used during tracking but is expensive to compute/query should be cached to keep tracking efficient. One example of such data are options that are stored in the piwik\_option table. Querying data for each tracking request means an extra unnecessary database query for each visitor action. Using a cache solves this problem.

### Example

```
public function setTrackerCacheGeneral(&$cacheContent)
{
    $cacheContent['MyPlugin.myCacheKey'] = Option::get('MyPlugin_myOption');
}
```

Callback Signature:

```
function(&$cacheContent)
```

- `array`

```
&$cacheContent
```

Array of cached data. Each piece of data must be mapped by name.

Usages:

[UserCountry::setTrackerCacheGeneral](#)

## Tracker.setVisitorIp

Defined in [Piwik/Tracker/Visit](#) in line [101](#)

Triggered after visits are tested for exclusion so plugins can modify the IP address persisted with a visit. This event is primarily used by the **AnonymizeIP** plugin to anonymize IP addresses.

Callback Signature:

```
function(&$this->visitorInfo['location_ip'])
```

- `string`

```
$ip
```

The visitor's IP address.

Usages:

[AnonymizeIP::setVisitorIpAddress](#)

## Translate

- [Translate.getClientSideTranslationKeys](#)

## Translate.getClientSideTranslationKeys

Defined in [Piwik/Translate](#) in line [196](#)

Triggered before generating the JavaScript code that allows i18n strings to be used in the browser. Plugins should subscribe to this event to specify which translations should be available to JavaScript.

Event handlers should add whole translation keys, ie, keys that include the plugin name.

### Example

```
public function getClientSideTranslationKeys(&$result)
{
    $result[] = "MyPlugin_MyTranslation";
}
```

Callback Signature:

```
function(&$result)
```

- array

```
&$result
```

The whole list of client side translation keys.

Usages:

[CoreHome::getClientSideTranslationKeys](#), [CorePluginsAdmin::getClientSideTranslationKeys](#), [CoreVisualizations::getClientSideTranslationKeys](#), [Dashboard::getClientSideTranslationKeys](#), [Feedback::getClientSideTranslationKeys](#), [Goals::getClientSideTranslationKeys](#), [Live::getClientSideTranslationKeys](#), [Overlay::getClientSideTranslationKeys](#), [ScheduledReports::getClientSideTranslationKeys](#), [SitesManager::getClientSideTranslationKeys](#), [Transitions::getClientSideTranslationKeys](#), [UserCountry::getClientSideTranslationKeys](#), [UsersManager::getClientSideTranslationKeys](#), [Widgetize::getClientSideTranslationKeys](#)

## Updater

- [Updater.checkForUpdates](#)

### Updater.checkForUpdates

Defined in [Piwik/FrontController](#) in line 409

Triggered after the platform is initialized and after the user has been authenticated, but before the platform has handled the request. Piwik uses this event to check for updates to Piwik.

Usages:

[CoreUpdater::updateCheck](#)

## User

- [User.getLanguage](#)
- [User.isNotAuthorized](#)

### User.getLanguage

Defined in [Piwik/Translate](#) in line 124

Triggered when the current user's language is requested. By default the current language is determined by the **language** query parameter. Plugins can override this logic by subscribing to this event.

#### Example

```
public function getLanguage(&$lang)
{
    $client = new My3rdPartyAPIClient();
    $thirdPartyLang = $client->getLanguageForUser(Piwik::getCurrentUserLogin());

    if (!empty($thirdPartyLang)) {
        $lang = $thirdPartyLang;
    }
}
```

Callback Signature:

`function(&$lang)`

- `string`

```
&$lang
```

The language that should be used for the current user. Will be initialized to the value of the **language** query parameter.

Usages:

[LanguagesManager::getLanguageToLoad](#)

## User.isNotAuthorized

Defined in [Piwik/FrontController](#) in line [156](#)

Triggered when a user with insufficient access permissions tries to view some resource. This event can be used to customize the error that occurs when a user is denied access (for example, displaying an error message, redirecting to a page other than login, etc.).

Callback Signature:

```
function($exception)
```

- [\NoAccessException](#)

```
$exception
```

The exception that was caught.

Usages:

[Login::noAccess](#)

## UsersManager

- [UsersManager.addUser.end](#)
- [UsersManager.deleteUser](#)
- [UsersManager.updateUser.end](#)

## UsersManager.addUser.end

Defined in [Piwik/Plugins/UsersManager/API](#) in line [406](#)

Triggered after a new user is created.

Callback Signature:

```
function($userLogin)
```

- `string`

```
$userLogin
```

The new user's login handle.

## UsersManager.deleteUser

Defined in [Piwik/Plugins/UsersManager/API](#) in line 661

Triggered after a user has been deleted. This event should be used to clean up any data that is related to the now deleted user. The **Dashboard** plugin, for example, uses this event to remove the user's dashboards.

Callback Signature:

```
function($userLogin)
```

- `string`

```
$userLogin
```

The login handle of the deleted user.

Usages:

[CoreAdminHome::cleanupUser](#), [Dashboard::deleteDashboardLayout](#), [LanguagesManager::deleteUserLanguage](#), [ScheduledReports::deleteUserReport](#)

## UsersManager.updateUser.end

Defined in [Piwik/Plugins/UsersManager/API](#) in line 468

Triggered after an existing user has been updated.

Callback Signature:

```
function($userLogin)
```

- `string`

```
$userLogin
```

The user's login handle.

## View

- [View.ReportsByDimension.render](#)

### View.ReportsByDimension.render

Defined in [Piwik\View\ReportsByDimension](#) in line [101](#)

Triggered before rendering ReportsByDimension views. Plugins can use this event to configure ReportsByDimension instances by adding or removing reports to display.

Callback Signature:

```
function($this)
```

- `ReportsByDimension`

```
$this
```

The view instance.

## ViewDataTable

- [ViewDataTable.addViewDataTable](#)
- [ViewDataTable.configure](#)
- [ViewDataTable.getDefaultType](#)

### ViewDataTable.addViewDataTable

Defined in [Piwik\ViewDataTable\Manager](#) in line [85](#)

Triggered when gathering all available DataTable visualizations. Plugins that want to expose new DataTable visualizations should subscribe to this event and add visualization class names to the incoming array.

## Example

```
public function addViewDataTable(&$visualizations)
{
    $visualizations[] = 'Piwik\\Plugins\\MyPlugin\\MyVisualiz
ion';
}
```

Callback Signature:

```
function(&$visualizations)
• array
```

```
&$visualizations
```

The array of all available visualizations.

Usages:

[CoreVisualizations::getAvailableDataTableVisualizations](#), [ExampleVisualization::getAvaila
bleVisualizations](#), [Goals::getAvailableDataTableVisualizations](#)

## ViewDataTable.configure

Defined in [Piwik\Plugin\ViewDataTable](#) in line [220](#)

Triggered during [ViewDataTable](#) construction. Subscribers should customize the view based on the report that is being displayed.

Plugins that define their own reports must subscribe to this event in order to specify how the Piwik UI should display the report.

## Example

```
// event handler

public function configureViewDataTable(ViewDataTable $view)
{
    switch ($view->requestConfig->apiMethodToRequestDataTable)
    {
        case 'VisitTime.getVisitInformationPerServerTime':
```

```
$view->config->enable_sort = true;  
$view->requestConfig->filter_limit = 10;  
break;  
}  
}
```

Callback Signature:

```
function($this)  
• ViewDataTable
```

```
$view
```

The instance to configure.

Usages:

[Actions::configureViewDataTable](#), [CustomVariables::configureViewDataTable](#), [DBStats::configureViewDataTable](#), [DevicesDetection::configureViewDataTable](#), [Goals::configureViewDataTable](#), [Provider::configureViewDataTable](#), [Referrers::configureViewDataTable](#), [UserCountry::configureViewDataTable](#), [UserSettings::configureViewDataTable](#), [VisitTime::configureViewDataTable](#), [VisitorInterest::configureViewDataTable](#)

## ViewDataTable.getDefaultType

Defined in [Piwik\ViewDataTable\Factory](#) in line [173](#)

Triggered when gathering the default view types for all available reports. If you define your own report, you may want to subscribe to this event to make sure the correct default Visualization is used (for example, a pie graph, bar graph, or something else).

If there is no default type associated with a report, the **table** visualization used.

### Example

```
public function getDefaultTypeViewDataTable(&$defaultViewTypes)  
{  
    $defaultViewTypes['Referrers.getSocials'] = HtmlTable::ID;  
    $defaultViewTypes['Referrers.getUrlForSocial'] = Pie::ID;
```

```
}
```

Callback Signature:

```
function(&self::$defaultViewTypes)
```

- array

```
$defaultViewTypes
```

The array mapping report IDs with visualization IDs.

Usages:

[DBStats::getDefaultTypeViewDataTable](#), [Live::getDefaultTypeViewDataTable](#), [Referrers::getDefaultTypeViewDataTable](#), [UserSettings::getDefaultTypeViewDataTable](#), [VisitTime::getDefaultTypeViewDataTable](#), [VisitorInterest::getDefaultTypeViewDataTable](#)

## WidgetsList

- [WidgetsList.addWidget](#)

### WidgetsList.addWidget

Defined in [Piwik/WidgetsList](#) in line 90

Used to collect all available dashboard widgets. Subscribe to this event to make your plugin's reports or other controller actions available as dashboard widgets. Event handlers should call the [WidgetsList::add\(\)](#) method for each new dashboard widget.

#### Example

```
public function addWidgets()
{
    WidgetsList::add('General_Actions', 'General_Pages', 'Actions', 'getPageUrls');
}
```

Usages:

[Actions::addWidget](#), [CoreHome::addWidget](#), [CustomVariables::addWidget](#), [DevicesDetection::addWidget](#), [ExampleRssWidget::addWidget](#), [Goals::addWidget](#), [Live::addWidget](#), [Provider::addWidget](#), [Referrers::addWidget](#), [SEO::addWidget](#), [UserCountry::addWidget](#)

[gets](#), [UserSettings::addWidget](#), [VisitFrequency::addWidget](#), [VisitTime::addWidget](#), [VisitorInterest::addWidget](#), [VisitsSummary::addWidget](#)

# Index

---

## A

•

`Archive`

— *Class in namespace*

`Piwik`

•

`ArchiveProcessor`

— *Class in namespace*

`Piwik`

•

`aggregateDataTableRecords()`

— *Method in class*

`ArchiveProcessor`

•

`aggregateNumericMetrics()`

— *Method in class*

`ArchiveProcessor`

- 

<code>applyQueuedFilters()</code>
-----------------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>addDataTable()</code>
-----------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>addRow()</code>
-----------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>addSummaryRow()</code>
------------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>addRowFromArray()</code>
--------------------------------

— *Method in class*

**DataTable**

- 

**addRowFromSimpleArray()**

— *Method in class*

**DataTable**

- 

**addRowsFromSerializedArray()**

— *Method in class*

**DataTable**

- 

**addRowsFromArray()**

— *Method in class*

**DataTable**

- 

**addRowsFromSimpleArray()**

— *Method in class*

**DataTable**

-

AddColumnsProcessedMetrics

— *Class in namespace*

Piwik\DataTable\Filter

•

AddColumnsProcessedMetricsGoal

— *Class in namespace*

Piwik\DataTable\Filter

•

AddSummaryRow

— *Class in namespace*

Piwik\DataTable\Filter

•

appendPercentSign()

— *Method in class*

CalculateEvolutionFilter

•

applyQueuedFilters()

— *Method in class*

Map

•

addTable()

— Method in class

Map

•

`addDataTable()`

— Method in class

Map

•

`addSubtable()`

— Method in class

Row

•

`addColumn()`

— Method in class

Row

•

`addColumns()`

— Method in class

Row

- 

<code>addMetadata()</code>
----------------------------

— *Method in class*

<code>Row</code>
------------------

- 

<code>addRowsFromArray()</code>
---------------------------------

— *Method in class*

<code>Simple</code>
---------------------

- 

<code>adjustForTimezone()</code>
----------------------------------

— *Method in class*

<code>Date</code>
-------------------

- 

<code>addDay()</code>
-----------------------

— *Method in class*

<code>Date</code>
-------------------

- 

<code>addHour()</code>
------------------------

— *Method in class*

Date

- 

`addHourTo()`

— *Method in class*

Date

- 

`addPeriod()`

— *Method in class*

Date

- 

`add()`

— *Method in class*

MenuAbstract

- 

`addEntry()`

— *Method in class*

MenuAdmin

-

`addEntry()`

— *Method in class*

`MenuTop`

•

`addAction()`

— *Method in class*

`Piwik`

•

`activate()`

— *Method in class*

`Plugin`

•

`API`

— *Class in namespace*

`Piwik\Plugin`

•

`Archiver`

— *Class in namespace*

`Piwik\Plugin`

- 

```
aggregateDayReport()
```

— *Method in class*

```
Archiver
```

- 

```
aggregateMultipleReports()
```

— *Method in class*

```
Archiver
```

- 

```
assignTemplateVar()
```

— *Method in class*

```
Visualization
```

- 

```
Archiver
```

— *Class in namespace*

```
Piwik\Plugins\Actions
```

- 

```
AllColumns
```

— *Class in namespace*

Piwik\Plugins\CoreVisualizations\Visualizations\HtmlTable

- 

Archiver

— *Class in namespace*

Piwik\Plugins\CustomVariables

- 

Archiver

— *Class in namespace*

Piwik\Plugins\DevicesDetection

- 

Archiver

— *Class in namespace*

Piwik\Plugins\Goals

- 

Archiver

— *Class in namespace*

Piwik\Plugins\Provider

- 

Archiver

— *Class in namespace*

Piwik\Plugins\Referrers

-

Archiver

— Class in namespace

Piwik\Plugins\UserCountry

•

Archiver

— Class in namespace

Piwik\Plugins\UserSettings

•

Archiver

— Class in namespace

Piwik\Plugins\VisitTime

•

Archiver

— Class in namespace

Piwik\Plugins\VisitorInterest

•

addLabelColumn()

— Method in class

RankingQuery

•

addColumn()

— Method in class

RankingQuery

- 

\$availableValues

— *Property in class*

Setting

- 

addPropertiesThatShouldBeAvailableClientSide()

— *Method in class*

Config

- 

addPropertiesThatCanBeOverwrittenByQueryParams()

— *Method in class*

Config

- 

addRelatedReport()

— *Method in class*

Config

-

```
addRelatedReports()
```

— *Method in class*

```
Config
```

•

```
addTranslation()
```

— *Method in class*

```
Config
```

•

```
addTranslations()
```

— *Method in class*

```
Config
```

•

```
$apiMethodToRequestDataTable
```

— *Property in class*

```
RequestConfig
```

•

```
addPropertiesThatShouldBeAvailableClientSide()
```

— *Method in class*

```
RequestConfig
```

- 

```
addPropertiesThatCanBeOverwrittenByqueryParams()
```

— *Method in class*

```
RequestConfig
```

- 

```
areQueuedFiltersDisabled()
```

— *Method in class*

```
RequestConfig
```

- 

```
areGenericFiltersDisabled()
```

— *Method in class*

```
RequestConfig
```

- 

```
add()
```

— *Method in class*

```
WidgetsList
```

# B

-

`build()`

— *Method in class*

`Archive`

•

`BaseFilter`

— *Class in namespace*

`Piwik\DataTable`

•

`BeautifyRangeLabels`

— *Class in namespace*

`Piwik\DataTable\Filter`

•

`beautify()`

— *Method in class*

`BeautifyRangeLabels`

•

`BeautifyTimeRangeLabels`

— *Class in namespace*

`Piwik\DataTable\Filter`

•

Bar

— Class in namespace

Piwik\Plugins\CoreVisualizations\Visualizations\JqplotGra  
ph

•

build()

— Method in class

Factory

C

•

Common

— Class in namespace

Piwik

•

Config

— Class in namespace

Piwik

•

CalculateEvolutionFilter

— Class in namespace

Piwik\DataTable\Filter

- 

`calculate()`

— *Method in class*

`CalculateEvolutionFilter`

- 

`ColumnCallbackAddColumn`

— *Class in namespace*

`Piwik\DataTable\Filter`

- 

`ColumnCallbackAddColumnPercentage`

— *Class in namespace*

`Piwik\DataTable\Filter`

- 

`ColumnCallbackAddColumnQuotient`

— *Class in namespace*

`Piwik\DataTable\Filter`

- 

`ColumnCallbackAddMetadata`

— *Class in namespace*

`Piwik\DataTable\Filter`

- 

`ColumnCallbackDeleteRow`

— Class in namespace

Piwik\DataTable\Filter

•

ColumnDelete

— Class in namespace

Piwik\DataTable\Filter

•

\$c

— Property in class

Row

•

compareWeek()

— Method in class

Date

•

compareMonth()

— Method in class

Date

•

compareYear()

— Method in class

Date

•

createDatabaseObject()

— Method in class

Db

•

copy()

— Method in class

Filesystem

•

copyRecursive()

— Method in class

Filesystem

•

\$context

— Property in class

Notification

- 

checkUserIsSuperUserOrTheUser()
---------------------------------

— *Method in class*

Piwik
-------

- 

checkUserIsNotAnonymous()
---------------------------

— *Method in class*

Piwik
-------

- 

checkUserIsSuperUser()
------------------------

— *Method in class*

Piwik
-------

- 

checkUserHasAdminAccess()
---------------------------

— *Method in class*

Piwik
-------

- 

checkUserHasSomeAdminAccess()
-------------------------------

— *Method in class*

Piwik

- 

checkUserHasViewAccess()

— *Method in class*

Piwik

- 

checkUserHasSomeViewAccess()

— *Method in class*

Piwik

- 

Controller

— *Class in namespace*

Piwik\Plugin

- 

checkTokenInUrl()

— *Method in class*

Controller

-

`$config`

— *Property in class*

`ViewDataTable`

•

`canDisplayViewDataTable()`

— *Method in class*

`ViewDataTable`

•

`Cloud`

— *Class in namespace*

`Piwik\Plugins\CoreVisualizations\Visualizations`

•

`$config`

— *Property in class*

`Cloud`

•

`$config`

— *Property in class*

`Graph`

- 

`$config`

— *Property in class*

`HtmlTable`

- 

`$config`

— *Property in class*

`JqplotGraph`

- 

`$config`

— *Property in class*

`Evolution`

- 

`canBeDisplayedForCurrentUser()`

— *Method in class*

`Setting`

- 

`clearCache()`

— *Method in class*

- - Site**
- - Config**
    - *Class in namespace*
- - Piwik\ViewDataTable**
- - \$clientSideProperties**
    - *Property in class*
- - Config**
- - \$columns\_to\_display**
    - *Property in class*
- - Config**
- - \$custom\_parameters**
    - *Property in class*
- - Config**
- - \$controllerName**

— *Property in class*

Config

•

\$controllerAction

— *Property in class*

Config

•

\$clientSideParameters

— *Property in class*

RequestConfig

## D

•

destroy()

— *Method in class*

Common

•

DataTable

— *Class in namespace*

Piwik

- 

<code>deleteColumn()</code>
-----------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>deleteColumns()</code>
------------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>deleteRow()</code>
--------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>deleteRowsOffset()</code>
---------------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>deleteRows()</code>
---------------------------

— *Method in class*

DataTable

- 

deleteColumns()

— *Method in class*

Map

- 

deleteRow()

— *Method in class*

Map

- 

deleteColumn()

— *Method in class*

Map

- 

deleteColumn()

— *Method in class*

Row

-

```
deleteMetadata()
```

— *Method in class*

```
Row
```

•

```
Date
```

— *Class in namespace*

```
Piwik
```

•

```
Db
```

— *Class in namespace*

```
Piwik
```

•

```
deleteAllRows()
```

— *Method in class*

```
Db
```

•

```
dropTables()
```

— *Method in class*

```
Db
```

•

`dispatch()`

— *Method in class*

`FrontController`

•

`downloadChunk()`

— *Method in class*

`Http`

•

`debug()`

— *Method in class*

`Log`

•

`discardNonce()`

— *Method in class*

`Nonce`

•

`delete()`

— *Method in class*

`Option`

- 

```
deleteLike()
```

— *Method in class*

```
Option
```

- 

```
deactivate()
```

— *Method in class*

```
Plugin
```

- 

```
$date
```

— *Property in class*

```
Controller
```

- 

```
Daily
```

— *Class in namespace*

```
Piwik\ScheduledTime
```

- 

```
$description
```

— *Property in class*

Setting

•

\$defaultValue

— *Property in class*

Setting

•

\$documentation

— *Property in class*

Config

•

\$datatable\_css\_class

— *Property in class*

Config

•

\$datatable\_js\_type

— *Property in class*

Config

- 

```
$disable_generic_filters
```

— *Property in class*

```
RequestConfig
```

- 

```
$disable_queued_filters
```

— *Property in class*

```
RequestConfig
```

## E

- 

```
enableRecursiveSort()
```

— *Method in class*

```
DataTable
```

- 

```
enableRecursiveFilters()
```

— *Method in class*

```
DataTable
```

- 

```
enableRecursive()
```

— Method in class

BaseFilter

•

ExcludeLowPopulation

— Class in namespace

Piwik\DataTable\Filter

•

enableRecursiveSort()

— Method in class

Map

•

exec()

— Method in class

Db

•

error()

— Method in class

Log

•

## Evolution

— *Class in namespace*

Piwik\Plugins\CoreVisualizations\Visualizations\JqplotGra  
ph

- 

## execute()

— *Method in class*

RankingQuery

- 

## \$enable\_sort

— *Property in class*

Config

- 

## \$export\_limit

— *Property in class*

Config

# F

- 

## factory()

— Method in class

Archive

•

forceSave()

— Method in class

Config

•

filter()

— Method in class

DataTable

•

fromSerializedArray()

— Method in class

DataTable

•

filter()

— Method in class

BaseFilter

- 

```
filterSubTable()
```

— *Method in class*

```
BaseFilter
```

- 

```
filter()
```

— *Method in class*

```
AddColumnsProcessedMetrics
```

- 

```
filter()
```

— *Method in class*

```
AddColumnsProcessedMetricsGoal
```

- 

```
filter()
```

— *Method in class*

```
AddSummaryRow
```

- 

```
filter()
```

— *Method in class*

ColumnCallbackAddColumn

- 

filter()

— Method in class

ColumnCallbackAddColumnQuotient

- 

filter()

— Method in class

ColumnCallbackAddMetadata

- 

filter()

— Method in class

ColumnCallbackDeleteRow

- 

filter()

— Method in class

ColumnDelete

-

`filter()`

— *Method in class*

`ExcludeLowPopulation`

•

`filter()`

— *Method in class*

`GroupBy`

•

`filter()`

— *Method in class*

`Limit`

•

`filter()`

— *Method in class*

`MetadataCallbackAddMetadata`

•

`filter()`

— *Method in class*

`Pattern`

- 

```
filter()
```

— *Method in class*

```
PatternRecursive
```

- 

```
filter()
```

— *Method in class*

```
ReplaceColumnNames
```

- 

```
filter()
```

— *Method in class*

```
ReplaceSummaryRowLabel
```

- 

```
filter()
```

— *Method in class*

```
Sort
```

- 

```
filter()
```

— Method in class

Truncate

•

filter()

— Method in class

Map

•

factory()

— Method in class

Date

•

fetchAll()

— Method in class

Db

•

fetchRow()

— Method in class

Db

- 

fetchOne()
------------

— *Method in class*

Db
----

- 

fetchAssoc()
--------------

— *Method in class*

Db
----

- 

Filesystem
------------

— *Class in namespace*

Piwik
-------

- 

FrontController
-----------------

— *Class in namespace*

Piwik
-------

- 

fetchRemoteFile()
-------------------

— *Method in class*

Http
------

- 

`$flags`

— *Property in class*

`Notification`

- 

`factory()`

— *Method in class*

`Period`

- 

`factory()`

— *Method in class*

`ScheduledTime`

- 

`$footer_icons`

— *Property in class*

`Config`

- 

`$filters`

— *Property in class*

Config

- 

Factory

— *Class in namespace*

Piwik\ViewDataTable

- 

\$filter\_sort\_column

— *Property in class*

RequestConfig

- 

\$filter\_sort\_order

— *Property in class*

RequestConfig

- 

\$filter\_limit

— *Property in class*

RequestConfig

- 

\$filter\_offset

— *Property in class*

RequestConfig
---------------

•

\$filter_pattern
------------------

— *Property in class*

RequestConfig
---------------

•

\$filter_column
-----------------

— *Property in class*

RequestConfig
---------------

•

\$filter_excludelowpop
------------------------

— *Property in class*

RequestConfig
---------------

•

\$filter_excludelowpop_value
------------------------------

— *Property in class*

RequestConfig
---------------

# G

•

<code>getRequestArrayFromString()</code>
------------------------------------------

— *Method in class*

<code>Request</code>
----------------------

•

<code>getClassNameAPI()</code>
--------------------------------

— *Method in class*

<code>Request</code>
----------------------

•

<code>getRequestParametersGET()</code>
----------------------------------------

— *Method in class*

<code>Request</code>
----------------------

•

<code>getBaseReportUrl()</code>
---------------------------------

— *Method in class*

<code>Request</code>
----------------------

•

<code>getCurrentUrlWithoutGenericFilters()</code>
---------------------------------------------------

— Method in class

Request

•

getRawSegmentFromRequest()

— Method in class

Request

•

getNumeric()

— Method in class

Archive

•

getBlob()

— Method in class

Archive

•

getDataTableFromNumeric()

— Method in class

Archive

- 

<code>getDataTable()</code>
-----------------------------

— *Method in class*

<b>Archive</b>
----------------

- 

<code>getdataTableExpanded()</code>
-------------------------------------

— *Method in class*

<b>Archive</b>
----------------

- 

<code>getParams()</code>
--------------------------

— *Method in class*

<b>Archive</b>
----------------

- 

<code>getdataTableFromArchive()</code>
----------------------------------------

— *Method in class*

<b>Archive</b>
----------------

- 

<code>getParams()</code>
--------------------------

— *Method in class*

ArchiveProcessor

- 

`getLogAggregator()`

— *Method in class*

ArchiveProcessor

- 

`getPeriod()`

— *Method in class*

Parameters

- 

`getSite()`

— *Method in class*

Parameters

- 

`getSegment()`

— *Method in class*

Parameters

-

```
getRequestParam()
```

— *Method in class*

```
Common
```

•

```
getLanguagesList()
```

— *Method in class*

```
Common
```

•

```
getLanguageToCountryList()
```

— *Method in class*

```
Common
```

•

```
getSqlStringFieldsArray()
```

— *Method in class*

```
Common
```

•

```
getSelectsFromRangedColumn()
```

— *Method in class*

```
LogAggregator
```

- 

```
getSortedByColumnName()
```

— *Method in class*

```
DataTable
```

- 

```
getRowFromLabel()
```

— *Method in class*

```
DataTable
```

- 

```
getRowIdFromLabel()
```

— *Method in class*

```
DataTable
```

- 

```
getEmptyClone()
```

— *Method in class*

```
DataTable
```

- 

```
getRowFromId()
```

— Method in class

DataTable

•

getRowFromIdSubDataTable()

— Method in class

DataTable

•

getId()

— Method in class

DataTable

•

getRows()

— Method in class

DataTable

•

getColumn()

— Method in class

DataTable

- 

```
getColumnsStartingWith()
```

— *Method in class*

```
DataTable
```

- 

```
getColumns()
```

— *Method in class*

```
DataTable
```

- 

```
getRowsMetadata()
```

— *Method in class*

```
DataTable
```

- 

```
getRowsCount()
```

— *Method in class*

```
DataTable
```

- 

```
getFirstRow()
```

— *Method in class*

`DataTable`

- 

`getLastRow()`

— *Method in class*

`DataTable`

- 

`getRowsCountRecursive()`

— *Method in class*

`DataTable`

- 

`getSerialized()`

— *Method in class*

`DataTable`

- 

`getMetadata()`

— *Method in class*

`DataTable`

-

```
getAllTableMetadata()
```

— *Method in class*

```
DataTable
```

•

```
getSingleUnitLabel()
```

— *Method in class*

```
BeautifyRangeLabels
```

•

```
getRangeLabel()
```

— *Method in class*

```
BeautifyRangeLabels
```

•

```
getUnboundedLabel()
```

— *Method in class*

```
BeautifyRangeLabels
```

•

```
getSingleUnitLabel()
```

— *Method in class*

```
BeautifyTimeRangeLabels
```

•

```
getRangeLabel()
```

— *Method in class*

```
BeautifyTimeRangeLabels
```

•

```
getUnboundedLabel()
```

— *Method in class*

```
BeautifyTimeRangeLabels
```

•

```
GroupBy
```

— *Class in namespace*

```
Piwik\DataTable\Filter
```

•

```
getKeyName()
```

— *Method in class*

```
Map
```

•

```
getRowsCount()
```

— *Method in class*

Map

- 

`getDatatables()`

— *Method in class*

Map

- 

`getTable()`

— *Method in class*

Map

- 

`getFirstRow()`

— *Method in class*

Map

- 

`getLastRow()`

— *Method in class*

Map

-

`getColumn()`

— *Method in class*

Map

•

`getEmptyClone()`

— *Method in class*

Map

•

`getMetadataIntersectArray()`

— *Method in class*

Map

•

`getColumns()`

— *Method in class*

Map

•

`getColumn()`

— *Method in class*

Row

- 

<code>getMetadata()</code>
----------------------------

— *Method in class*

Row
-----

- 

<code>getColumns()</code>
---------------------------

— *Method in class*

Row
-----

- 

<code>getIdSubDataTable()</code>
----------------------------------

— *Method in class*

Row
-----

- 

<code>getSubtable()</code>
----------------------------

— *Method in class*

Row
-----

- 

<code>getDatetime()</code>
----------------------------

— Method in class

Date

•

getDateStartUTC()

— Method in class

Date

•

getDateEndUTC()

— Method in class

Date

•

getTimestampUTC()

— Method in class

Date

•

getTimestamp()

— Method in class

Date

- 

getLocalized()
----------------

— *Method in class*

Date
------

- 

get()
-------

— *Method in class*

Db
----

- 

getDbLock()
-------------

— *Method in class*

Db
----

- 

globr()
---------

— *Method in class*

Filesystem
------------

- 

getTransportMethod()
----------------------

— *Method in class*

Http

- 

getIPv4FromMappedIPv6()

— *Method in class*

IP

- 

getIpsForRange()

— *Method in class*

IP

- 

getIpFromHeader()

— *Method in class*

IP

- 

getNonProxyIpFromHeader()

— *Method in class*

IP

-

```
getLastIpFromList()
```

— *Method in class*

```
IP
```

•

```
getHostByAddr()
```

— *Method in class*

```
IP
```

•

```
getVisitsMetricNames()
```

— *Method in class*

```
Metrics
```

•

```
getMappingFromIdToName()
```

— *Method in class*

```
Metrics
```

•

```
getDefaultMetricTranslations()
```

— *Method in class*

```
Metrics
```

- 

```
getDefaultMetrics()
```

— *Method in class*

```
Metrics
```

- 

```
getDefaultProcessedMetrics()
```

— *Method in class*

```
Metrics
```

- 

```
getReadableColumnName()
```

— *Method in class*

```
Metrics
```

- 

```
getMetricIdsToProcessReportTotal()
```

— *Method in class*

```
Metrics
```

- 

```
getDefaultMetricsDocumentation()
```

— Method in class

Metrics

•

getPercentVisitColumn()

— Method in class

Metrics

•

getPrettyNumber()

— Method in class

MetricsFormatter

•

getPrettyTimeFromSeconds()

— Method in class

MetricsFormatter

•

getPrettySizeFromBytes()

— Method in class

MetricsFormatter

- 

```
getPrettyMoney()
```

— *Method in class*

```
MetricsFormatter
```

- 

```
getPrettyValue()
```

— *Method in class*

```
MetricsFormatter
```

- 

```
getCurrencySymbol()
```

— *Method in class*

```
MetricsFormatter
```

- 

```
getCurrencyList()
```

— *Method in class*

```
MetricsFormatter
```

- 

```
getNonce()
```

— *Method in class*

Nonce

- 

`getOrigin()`

— *Method in class*

Nonce

- 

`getAcceptableOrigins()`

— *Method in class*

Nonce

- 

`getPriority()`

— *Method in class*

Notification

- 

`get()`

— *Method in class*

Option

-

`getDateStart()`

— *Method in class*

`Period`

•

`getDateEnd()`

— *Method in class*

`Period`

•

`getId()`

— *Method in class*

`Period`

•

`getLabel()`

— *Method in class*

`Period`

•

`getNumberOfSubperiods()`

— *Method in class*

`Period`

- 

<code>getSubperiods()</code>
------------------------------

— *Method in class*

<code>Period</code>
---------------------

- 

<code>getPrettyString()</code>
--------------------------------

— *Method in class*

<code>Period</code>
---------------------

- 

<code>getLocalizedShortString()</code>
----------------------------------------

— *Method in class*

<code>Period</code>
---------------------

- 

<code>getLocalizedLongString()</code>
---------------------------------------

— *Method in class*

<code>Period</code>
---------------------

- 

<code>getRangeString()</code>
-------------------------------

— Method in class

Period

•

getLocalizedShortString()

— Method in class

Range

•

getLocalizedLongString()

— Method in class

Range

•

getDateStart()

— Method in class

Range

•

getPrettyString()

— Method in class

Range

- 

<code>getDateEnd()</code>
---------------------------

— *Method in class*

Range
-------

- 

<code>getLastDate()</code>
----------------------------

— *Method in class*

Range
-------

- 

<code>getRelativeToDate()</code>
----------------------------------

— *Method in class*

Range
-------

- 

<code>getCurrentUserEmail()</code>
------------------------------------

— *Method in class*

Piwik
-------

- 

<code>getSuperUserLogin()</code>
----------------------------------

— *Method in class*

Piwik

- 

`getSuperUserEmail()`

— *Method in class*

Piwik

- 

`getCurrentUserLogin()`

— *Method in class*

Piwik

- 

`getCurrentUserTokenAuth()`

— *Method in class*

Piwik

- 

`getInformation()`

— *Method in class*

Plugin

-

```
getListHooksRegistered()
```

— *Method in class*

```
Plugin
```

•

```
getVersion()
```

— *Method in class*

```
Plugin
```

•

```
getPluginName()
```

— *Method in class*

```
Plugin
```

•

```
getPluginNameFromBacktrace()
```

— *Method in class*

```
Plugin
```

•

```
getProcessor()
```

— *Method in class*

```
Archiver
```

- 

```
getLogAggregator()
```

— *Method in class*

```
Archiver
```

- 

```
getDateParameterInTimezone()
```

— *Method in class*

```
Controller
```

- 

```
getDefaultValue()
```

— *Method in class*

```
Controller
```

- 

```
getLastUnitGraph()
```

— *Method in class*

```
Controller
```

- 

```
getLastUnitGraphAcrossPlugins()
```

— Method in class

Controller
------------

•

getUrlSparkline()
-------------------

— Method in class

Controller
------------

•

getDefaultValueId()
---------------------

— Method in class

Controller
------------

•

getDefaultValue()
-------------------

— Method in class

Controller
------------

•

getDefaultValuePeriod()
-------------------------

— Method in class

Controller
------------

- 

```
getCalendarPrettyDate()
```

— *Method in class*

```
Controller
```

- 

```
getEvolutionHtml()
```

— *Method in class*

```
Controller
```

- 

```
getThemeEnabled()
```

— *Method in class*

```
Manager
```

- 

```
getInstalledPluginsName()
```

— *Method in class*

```
Manager
```

- 

```
getMissingPlugins()
```

— *Method in class*

Manager

- 

`getIntroduction()`

— *Method in class*

Settings

- 

`getSettingsForCurrentUser()`

— *Method in class*

Settings

- 

`getSettings()`

— *Method in class*

Settings

- 

`getSettingValue()`

— *Method in class*

Settings

-

```
getDefaultValue()
```

— *Method in class*

```
ViewDataTable
```

•

```
getDefaultRequestConfig()
```

— *Method in class*

```
ViewDataTable
```

•

```
getViewDataTableId()
```

— *Method in class*

```
ViewDataTable
```

•

```
getDataTable()
```

— *Method in class*

```
ViewDataTable
```

•

```
Graph
```

— *Class in namespace*

```
Piwik\Plugins\CoreVisualizations\Visualizations
```

- 

```
Goals
```

— *Class in namespace*

```
Piwik\Plugins\Goals\Visualizations
```

- 

```
generateQuery()
```

— *Method in class*

```
RankingQuery
```

- 

```
getObjectInstance()
```

— *Method in class*

```
ScheduledTask
```

- 

```
getClassName()
```

— *Method in class*

```
ScheduledTask
```

- 

```
getMethodName()
```

— *Method in class*

```
ScheduledTask
```

- 

<code>getMethodParameter()</code>
-----------------------------------

— *Method in class*

<code>ScheduledTask</code>
----------------------------

- 

<code>getScheduledTime()</code>
---------------------------------

— *Method in class*

<code>ScheduledTask</code>
----------------------------

- 

<code>getRescheduledTime()</code>
-----------------------------------

— *Method in class*

<code>ScheduledTask</code>
----------------------------

- 

<code>getPriority()</code>
----------------------------

— *Method in class*

<code>ScheduledTask</code>
----------------------------

- 

<code>getName()</code>
------------------------

— Method in class

ScheduledTask

•

getString()

— Method in class

Segment

•

getHash()

— Method in class

Segment

•

getSelectQuery()

— Method in class

Segment

•

getPiwikUrl()

— Method in class

SettingsPiwik

- 

getName()
-----------

— *Method in class*

Setting
---------

- 

getValue()
------------

— *Method in class*

Setting
---------

- 

getKey()
----------

— *Method in class*

Setting
---------

- 

getOrder()
------------

— *Method in class*

Setting
---------

- 

getOrder()
------------

— *Method in class*

**SystemSetting**

- 

**getOrder()**

— *Method in class*

**UserSetting**

- 

**getInstance()**

— *Method in class*

**Singleton**

- 

**getName()**

— *Method in class*

**Site**

- 

**getMainUrl()**

— *Method in class*

**Site**

-

`getId()`

— *Method in class*

`Site`

•

`getType()`

— *Method in class*

`Site`

•

`getCreationDate()`

— *Method in class*

`Site`

•

`getTimezone()`

— *Method in class*

`Site`

•

`getCurrency()`

— *Method in class*

`Site`

- 

```
getExcludedIps()
```

— *Method in class*

```
Site
```

- 

```
getExcludedQueryParameters()
```

— *Method in class*

```
Site
```

- 

```
getSearchKeywordParameters()
```

— *Method in class*

```
Site
```

- 

```
getSearchCategoryParameters()
```

— *Method in class*

```
Site
```

- 

```
getIdSitesFromIdSitesString()
```

— Method in class

Site

•

getNameFor()

— Method in class

Site

•

getTimezoneFor()

— Method in class

Site

•

getTypeFor()

— Method in class

Site

•

getCreationDateFor()

— Method in class

Site

- 

`getMainUrlFor()`

— *Method in class*

`Site`

- 

`getCurrencyFor()`

— *Method in class*

`Site`

- 

`getExcludedIpsFor()`

— *Method in class*

`Site`

- 

`getExcludedQueryParametersFor()`

— *Method in class*

`Site`

- 

`getCurrentUrl()`

— *Method in class*

Url

- 

getCurrentUrlWithoutQueryString()

— Method in class

Url

- 

getCurrentUrlWithoutFileName()

— Method in class

Url

- 

getCurrentScriptPath()

— Method in class

Url

- 

getCurrentScriptName()

— Method in class

Url

-

```
getCurrentScheme()
```

— *Method in class*

```
Url
```

•

```
getCurrentHost()
```

— *Method in class*

```
Url
```

•

```
getCurrentQueryString()
```

— *Method in class*

```
Url
```

•

```
getArrayFromCurrentQueryString()
```

— *Method in class*

```
Url
```

•

```
getCurrentQueryStringWithParametersModified()
```

— *Method in class*

```
Url
```

- 

getQueryStringFromParameters()
--------------------------------

— *Method in class*

Url
-----

- 

getReferrer()
---------------

— *Method in class*

Url
-----

- 

getQueryStringWithExcludedParameters()
----------------------------------------

— *Method in class*

UrlHelper
-----------

- 

getParseUrlReverse()
----------------------

— *Method in class*

UrlHelper
-----------

- 

getArrayFromQueryString()
---------------------------

— Method in class

UrlHelper

•

getParameterFromQueryString()

— Method in class

UrlHelper

•

getPathAndQueryFromUrl()

— Method in class

UrlHelper

•

getTemplateFile()

— Method in class

View

•

getTemplateVars()

— Method in class

View

- 

`getProperties()`

— *Method in class*

`Config`

- 

`getProperties()`

— *Method in class*

`RequestConfig`

- 

`getApiModuleToRequest()`

— *Method in class*

`RequestConfig`

- 

`getApiMethodToRequest()`

— *Method in class*

`RequestConfig`

- 

`get()`

— *Method in class*

WidgetsList

# H

•

Http

— Class in namespace

Piwik

•

hasNoClear()

— Method in class

Notification

•

HtmlTable

— Class in namespace

Piwik\Plugins\CoreVisualizations\Visualizations

•

Hourly

— Class in namespace

Piwik\ScheduledTime

•

\$hide\_annotations\_view

— *Property in class*

Config

|

•

insertNumericRecords()

— *Method in class*

ArchiveProcessor

•

insertNumericRecord()

— *Method in class*

ArchiveProcessor

•

insertBlobRecord()

— *Method in class*

ArchiveProcessor

•

isEqual()

— *Method in class*

DataTable

- 

<code>isSubtableLoaded()</code>
---------------------------------

— *Method in class*

Row
-----

- 

<code>isSummaryRow()</code>
-----------------------------

— *Method in class*

Row
-----

- 

<code>isEqual()</code>
------------------------

— *Method in class*

Row
-----

- 

<code>isLater()</code>
------------------------

— *Method in class*

Date
------

- 

<code>isEarlier()</code>
--------------------------

— Method in class

Date

•

isToday()

— Method in class

Date

•

isLockPrivilegeGranted()

— Method in class

Db

•

IP

— Class in namespace

Piwik

•

isIPv4()

— Method in class

IP

•

`isIPv6()`

— *Method in class*

`IP`

•

`isMappedIPv4()`

— *Method in class*

`IP`

•

`isIpInRange()`

— *Method in class*

`IP`

•

`info()`

— *Method in class*

`Log`

•

`isMultiplePeriod()`

— *Method in class*

`Period`

- 

<code>isUserIsSuperUserOrTheUser()</code>
-------------------------------------------

— *Method in class*

Piwik
-------

- 

<code>isUserIsSuperUser()</code>
----------------------------------

— *Method in class*

Piwik
-------

- 

<code>isUserIsAnonymous()</code>
----------------------------------

— *Method in class*

Piwik
-------

- 

<code>isUserHasAdminAccess()</code>
-------------------------------------

— *Method in class*

Piwik
-------

- 

<code>isUserHasSomeAdminAccess()</code>
-----------------------------------------

— Method in class

Piwik

•

`isUserHasViewAccess()`

— Method in class

Piwik

•

`isUserHasSomeViewAccess()`

— Method in class

Piwik

•

`isValidEmailString()`

— Method in class

Piwik

•

`install()`

— Method in class

Plugin

- 

```
isTheme()
```

— *Method in class*

```
Plugin
```

- 

```
$idSite
```

— *Property in class*

```
Controller
```

- 

```
isPluginActivated()
```

— *Method in class*

```
Manager
```

- 

```
isPluginLoaded()
```

— *Method in class*

```
Manager
```

- 

```
isViewDataTableId()
```

— *Method in class*

`ViewDataTable`

- 

`isRequestingSingleDataTable()`

— *Method in class*

`ViewDataTable`

- 

`isThereDataToDisplay()`

— *Method in class*

`Visualization`

- 

`isEmpty()`

— *Method in class*

`Segment`

- 

`isSegmentationEnabled()`

— *Method in class*

`SettingsPiwik`

-

```
isUniqueVisitorsEnabled()
```

— *Method in class*

```
SettingsPiwik
```

•

```
isArchivePhpTriggered()
```

— *Method in class*

```
SettingsServer
```

•

```
isIIS()
```

— *Method in class*

```
SettingsServer
```

•

```
isApache()
```

— *Method in class*

```
SettingsServer
```

•

```
isWindows()
```

— *Method in class*

```
SettingsServer
```

- 

```
isTimezoneSupportEnabled()
```

— *Method in class*

```
SettingsServer
```

- 

```
isGdExtensionEnabled()
```

— *Method in class*

```
SettingsServer
```

- 

```
$introduction
```

— *Property in class*

```
Setting
```

- 

```
$inlineHelp
```

— *Property in class*

```
Setting
```

- 

```
isEcommerceEnabled()
```

— Method in class

Site

•

isSiteSearchEnabled()

— Method in class

Site

•

isEcommerceEnabledFor()

— Method in class

Site

•

isSiteSearchEnabledFor()

— Method in class

Site

•

isLocalUrl()

— Method in class

Url

- 

`$idSubtable`

— *Property in class*

`RequestConfig`

- 

`isDefined()`

— *Method in class*

`WidgetsList`

## J

- 

`JqplotGraph`

— *Class in namespace*

`Piwik\Plugins\CoreVisualizations\Visualizations`

## L

- 

`LogAggregator`

— *Class in namespace*

`Piwik\DataAccess`

- 

`Limit`

— Class in namespace

Piwik\DataTable\Filter

•

\$lockPrivilegeGranted

— Property in class

Db

•

lockTables()

— Method in class

Db

•

long2ip()

— Method in class

IP

•

Log

— Class in namespace

Piwik

M

•

`mb_substr()`

— *Method in class*

`Common`

•

`mb_strlen()`

— *Method in class*

`Common`

•

`mb_strtolower()`

— *Method in class*

`Common`

•

`makeFromIndexedArray()`

— *Method in class*

`DataTable`

•

`mergeSubtables()`

— *Method in class*

`DataTable`

- 

<code>makeFromSimpleArray()</code>
------------------------------------

— *Method in class*

<code>DataTable</code>
------------------------

- 

<code>MetadataCallbackAddMetadata</code>
------------------------------------------

— *Class in namespace*

<code>Piwik\DataTable\Filter</code>
-------------------------------------

- 

<code>MetadataCallbackReplace</code>
--------------------------------------

— *Class in namespace*

<code>Piwik\DataTable\Filter</code>
-------------------------------------

- 

<code>Map</code>
------------------

— *Class in namespace*

<code>Piwik\DataTable</code>
------------------------------

- 

<code>mergeChildren()</code>
------------------------------

— *Method in class*

<code>Map</code>
------------------

- 

mergeSubtables()
------------------

— *Method in class*

Map
-----

- 

\$maxVisitsSummed
-------------------

— *Property in class*

Row
-----

- 

mkdir()
---------

— *Method in class*

Filesystem
------------

- 

Mail
------

— *Class in namespace*

Piwik
-------

- 

MenuAbstract
--------------

— *Class in namespace*

Piwik\Menu

•

MenuAdmin

— *Class in namespace*

Piwik\Menu

•

MenuMain

— *Class in namespace*

Piwik\Menu

•

MenuTop

— *Class in namespace*

Piwik\Menu

•

Metrics

— *Class in namespace*

Piwik

•

\$mappingFromIdToName

— *Property in class*

Metrics

•

`$mappingFromIdToNameGoal`

— *Property in class*

`Metrics`

•

`$mappingFromNameToId`

— *Property in class*

`Metrics`

•

`MetricsFormatter`

— *Class in namespace*

`Piwik`

•

`$message`

— *Property in class*

`Notification`

•

`Manager`

— *Class in namespace*

`Piwik\Notification`

- 

`makePeriodFromQueryParams()`

— *Method in class*

`Period`

- 

`Manager`

— *Class in namespace*

`Piwik\Plugin`

- 

`Monthly`

— *Class in namespace*

`Piwik\ScheduledTime`

- 

`$metrics_documentation`

— *Property in class*

`Config`

# N

- 

`numberSort()`

— Method in class

Sort

•

naturalSort()

— Method in class

Sort

•

now()

— Method in class

Date

•

N2P()

— Method in class

IP

•

NoAccessException

— Class in namespace

Piwik

•

Nonce

— *Class in namespace*

Piwik

•

Notification

— *Class in namespace*

Piwik

•

notify()

— *Method in class*

Manager

O

•

optimizeTables()

— *Method in class*

Db

•

Option

— *Class in namespace*

Piwik

•

`$overridableProperties`

— *Property in class*

`Config`

•

`$overridableProperties`

— *Property in class*

`RequestConfig`

P

•

`process()`

— *Method in class*

`Request`

•

`processRequest()`

— *Method in class*

`Request`

•

`Parameters`

— *Class in namespace*

Piwik\ArchiveProcessor

•

`prefixTable()`

— *Method in class*

Common

•

`prependPlusSignToNumber()`

— *Method in class*

CalculateEvolutionFilter

•

Pattern

— *Class in namespace*

Piwik\DataTable\Filter

•

PatternRecursive

— *Class in namespace*

Piwik\DataTable\Filter

•

`P2N()`

— *Method in class*

IP

- 

`prettyPrint()`

— *Method in class*

IP

- 

`$priority`

— *Property in class*

Notification

- 

Period

— *Class in namespace*

Piwik

- 

`parseDateRange()`

— *Method in class*

Range

- 

Piwik

— Class in namespace

Piwik

•

postEvent()

— Method in class

Piwik

•

Plugin

— Class in namespace

Piwik

•

postLoad()

— Method in class

Plugin

•

\$pluginName

— Property in class

Controller

•

Pie

— Class in namespace

```
Piwik\Plugins\CoreVisualizations\Visualizations\JqplotGra  
ph
```

- 

```
partitionResultIntoMultipleGroups()
```

— Method in class

```
RankingQuery
```

## Q

- 

```
queryVisitsByDimension()
```

— Method in class

```
LogAggregator
```

- 

```
queryEcommerceItems()
```

— Method in class

```
LogAggregator
```

- 

```
queryActionsByDimension()
```

— Method in class

LogAggregator

- 

queueFilter()

— Method in class

DataTable

- 

queueFilter()

— Method in class

Map

- 

query()

— Method in class

Db

# R

- 

Request

— Class in namespace

Piwik\API

-

`renameColumn()`

— *Method in class*

`DataTable`

•

`ReplaceColumnNames`

— *Class in namespace*

`Piwik\DataTable\Filter`

•

`ReplaceSummaryRowLabel`

— *Class in namespace*

`Piwik\DataTable\Filter`

•

`renameColumn()`

— *Method in class*

`Map`

•

`Row`

— *Class in namespace*

`Piwik\DataTable`

•

`renameColumn()`

— *Method in class*

`Row`

•

`removeSubtable()`

— *Method in class*

`Row`

•

`releaseDbLock()`

— *Method in class*

`Db`

•

`$raw`

— *Property in class*

`Notification`

•

`Range`

— *Class in namespace*

`Piwik\Period`

- 

redirectToModule()
--------------------

— *Method in class*

Piwik
-------

- 

renderReport()
----------------

— *Method in class*

Controller
------------

- 

redirectToIndex()
-------------------

— *Method in class*

Controller
------------

- 

returnLoadedPluginsInfo()
---------------------------

— *Method in class*

Manager
---------

- 

removeAllPluginSettings()
---------------------------

— Method in class

Settings

•

removeSettingValue()

— Method in class

Settings

•

\$requestConfig

— Property in class

ViewDataTable

•

render()

— Method in class

ViewDataTable

•

RankingQuery

— Class in namespace

Piwik

•

```
removeAllUserSettingsForUser()
```

— *Method in class*

```
UserSetting
```

•

```
rescheduleTask()
```

— *Method in class*

```
TaskScheduler
```

•

```
redirectToReferrer()
```

— *Method in class*

```
Url
```

•

```
redirectToUrl()
```

— *Method in class*

```
Url
```

•

```
render()
```

— *Method in class*

```
View
```

•

`$related_reports`

— *Property in class*

`Config`

•

`$report_id`

— *Property in class*

`Config`

•

`RequestConfig`

— *Class in namespace*

`Piwik\ViewDataTable`

•

`$request_parameters_to_modify`

— *Property in class*

`RequestConfig`

•

`remove()`

— *Method in class*

# S

•

`WidgetsList`

— *Method in class*

`Common`

•

`sort()`

— *Method in class*

`DataTable`

•

`setMaximumDepthLevelAllowedAtLeast()`

— *Method in class*

`DataTable`

•

`setMetadata()`

— *Method in class*

`DataTable`

- 

```
setMetadataValues()
```

— *Method in class*

```
DataTable
```

- 

```
setAllTableMetadata()
```

— *Method in class*

```
DataTable
```

- 

```
setMaximumAllowedRows()
```

— *Method in class*

```
DataTable
```

- 

```
Sort
```

— *Class in namespace*

```
Piwik\DataTable\Filter
```

- 

```
setOrder()
```

— *Method in class*

```
Sort
```

- 

<code>sortString()</code>
---------------------------

— *Method in class*

Sort
------

- 

<code>setKeyName()</code>
---------------------------

— *Method in class*

Map
-----

- 

<code>sumSubtable()</code>
----------------------------

— *Method in class*

Row
-----

- 

<code>setSubtable()</code>
----------------------------

— *Method in class*

Row
-----

- 

<code>setColumns()</code>
---------------------------

— Method in class

Row

•

`setColumn()`

— Method in class

Row

•

`setMetadata()`

— Method in class

Row

•

`sumRow()`

— Method in class

Row

•

`sumRowMetadata()`

— Method in class

Row

- 

Simple

— *Class in namespace*

Piwik\DataTable

- 

setTimezone()

— *Method in class*

Date

- 

setTime()

— *Method in class*

Date

- 

setDay()

— *Method in class*

Date

- 

setYear()

— *Method in class*

Date

- 

`subDay()`

— *Method in class*

Date

- 

`subWeek()`

— *Method in class*

Date

- 

`subMonth()`

— *Method in class*

Date

- 

`subYear()`

— *Method in class*

Date

-

`subHour()`

— *Method in class*

`Date`

•

`subPeriod()`

— *Method in class*

`Date`

•

`secondsToDays()`

— *Method in class*

`Date`

•

`segmentedFetchFirst()`

— *Method in class*

`Db`

•

`segmentedFetchOne()`

— *Method in class*

`Db`

- 

```
segmentedFetchAll()
```

— *Method in class*

```
Db
```

- 

```
segmentedQuery()
```

— *Method in class*

```
Db
```

- 

```
sendHttpRequest()
```

— *Method in class*

```
Http
```

- 

```
sanitizeIp()
```

— *Method in class*

```
IP
```

- 

```
sanitizeIpRange()
```

— Method in class

IP

•

setFrom()

— Method in class

Mail

•

set()

— Method in class

Option

•

setDefaultEndDate()

— Method in class

Range

•

\$strDate

— Property in class

Controller

- 

```
$site
```

— *Property in class*

```
Controller
```

- 

```
setDate()
```

— *Method in class*

```
Controller
```

- 

```
setMinDateView()
```

— *Method in class*

```
Controller
```

- 

```
setMaxDateView()
```

— *Method in class*

```
Controller
```

- 

```
setGeneralVariablesView()
```

— *Method in class*

Controller

- 

`setBasicVariablesView()`

— *Method in class*

Controller

- 

`setHostValidationVariablesView()`

— *Method in class*

Controller

- 

`setPeriodVariablesView()`

— *Method in class*

Controller

- 

`Settings`

— *Class in namespace*

Piwik\Plugin

-

`save()`

— *Method in class*

`Settings`

•

`setSettingValue()`

— *Method in class*

`Settings`

•

`setDataTable()`

— *Method in class*

`ViewDataTable`

•

`SimpleTable`

— *Class in namespace*

`Piwik\Plugins\ExampleVisualization`

•

`setLimit()`

— *Method in class*

`RankingQuery`

- 

`setOthersLabel()`

— *Method in class*

`RankingQuery`

- 

`setColumnToMarkExcludedRows()`

— *Method in class*

`RankingQuery`

- 

`ScheduledTask`

— *Class in namespace*

`Piwik`

- 

`ScheduledTime`

— *Class in namespace*

`Piwik`

- 

`setHour()`

— *Method in class*

`ScheduledTime`

•

Segment

— *Class in namespace*

Piwik

•

SettingsPiwik

— *Class in namespace*

Piwik

•

SettingsServer

— *Class in namespace*

Piwik

•

Setting

— *Class in namespace*

Piwik\Settings

•

setStorage()

— *Method in class*

Setting

•

setValue()

— Method in class

Setting

•

SystemSetting

— Class in namespace

Piwik\Settings

•

setUserLogin()

— Method in class

UserSetting

•

Singleton

— Class in namespace

Piwik

•

Site

— Class in namespace

Piwik

•

setSites()

— Method in class

Site

- 

`setSitesFromArray()`

— *Method in class*

Site

- 

`setContent-Type()`

— *Method in class*

View

- 

`setXFrameOptions()`

— *Method in class*

View

- 

`singleReport()`

— *Method in class*

View

-

```
$show_visualization_only
```

— *Property in class*

```
Config
```

•

```
$show_goals
```

— *Property in class*

```
Config
```

•

```
$show_exclude_low_population
```

— *Property in class*

```
Config
```

•

```
$show_flatten_table
```

— *Property in class*

```
Config
```

•

```
$show_table
```

— *Property in class*

```
Config
```

- 

<code>\$show_table_all_columns</code>
---------------------------------------

— *Property in class*

Config
--------

- 

<code>\$show_footer</code>
----------------------------

— *Property in class*

Config
--------

- 

<code>\$show_footer_icons</code>
----------------------------------

— *Property in class*

Config
--------

- 

<code>\$show_all_views_icons</code>
-------------------------------------

— *Property in class*

Config
--------

- 

<code>\$show_active_view_icon</code>
--------------------------------------

— *Property in class*

Config
--------

•

\$show_related_reports
------------------------

— *Property in class*

Config
--------

•

\$show_limit_control
----------------------

— *Property in class*

Config
--------

•

\$show_search
---------------

— *Property in class*

Config
--------

•

\$show_bar_chart
------------------

— *Property in class*

Config
--------

- 

\$show_pie_chart
------------------

— *Property in class*

Config
--------

- 

\$show_tag_cloud
------------------

— *Property in class*

Config
--------

- 

\$show_export_as_rss_feed
---------------------------

— *Property in class*

Config
--------

- 

\$show_ecommerce
------------------

— *Property in class*

Config
--------

- 

\$show_footer_message
-----------------------

— *Property in class*

Config

- 

`$self_url`

— *Property in class*

Config

- 

`$search_recursive`

— *Property in class*

Config

- 

`$show_export_as_image_icon`

— *Property in class*

Config

- 

`$subtable_controller_action`

— *Property in class*

Config

-

`$show_pagination_control`

— *Property in class*

`Config`

•

`$show_offset_information`

— *Property in class*

`Config`

•

`setDefaultSort()`

— *Method in class*

`RequestConfig`

T

•

`Truncate`

— *Class in namespace*

`Piwik\DataTable\Filter`

•

`toString()`

— *Method in class*

`Date`

- 

```
today()
```

— *Method in class*

```
Date
```

- 

```
$title
```

— *Property in class*

```
Notification
```

- 

```
$type
```

— *Property in class*

```
Notification
```

- 

```
toString()
```

— *Method in class*

```
Period
```

- 

```
translate()
```

— Method in class

Piwik

•

\$type

— Property in class

Setting

•

\$transform

— Property in class

Setting

•

\$title

— Property in class

Setting

•

TaskScheduler

— Class in namespace

Piwik

•

`$translations`

— *Property in class*

`Config`

•

`$title`

— *Property in class*

`Config`

•

`$tooltip_metadata_name`

— *Property in class*

`Config`

U

•

`unprefixTable()`

— *Method in class*

`Common`

•

`unsanitizeInputValues()`

— *Method in class*

Common

- 

`unlockAllTables()`

— *Method in class*

Db

- 

`unlinkRecursive()`

— *Method in class*

Filesystem

- 

`uninstall()`

— *Method in class*

Plugin

- 

`$uiControlType`

— *Property in class*

Setting

-

`$uiControlAttributes`

— *Property in class*

`Setting`

•

`UserSetting`

— *Class in namespace*

`Piwik\Settings`

•

`Url`

— *Class in namespace*

`Piwik`

•

`UrlHelper`

— *Class in namespace*

`Piwik`

V

•

`verbose()`

— *Method in class*

`Log`

•

`verifyNonce()`

— *Method in class*

`Nonce`

•

`ViewDataTable`

— *Class in namespace*

`Piwik\Plugin`

•

`Visualization`

— *Class in namespace*

`Piwik\Plugin`

•

`VisitorLog`

— *Class in namespace*

`Piwik\Plugins\Live`

•

`$validate`

— *Property in class*

`Setting`

•

Version

— Class in namespace

Piwik

•

View

— Class in namespace

Piwik

W

•

walkPath()

— Method in class

DataTable

•

warning()

— Method in class

Log

•

Weekly

— Class in namespace

Piwik\ScheduledTime

•

WidgetsList

— *Class in namespace*

Piwik

Y

•

yesterday()

— *Method in class*

Date

•

yesterdaySameTime()

— *Method in class*

Date

•

\$y\_axis\_unit

— *Property in class*

Config

—

•

`__construct()`

— *Method in class*

`Request`

•

`__get()`

— *Method in class*

`Config`

•

`__set()`

— *Method in class*

`Config`

•

`__construct()`

— *Method in class*

`DataTable`

•

`__destruct()`

— Method in class

```
DataTable
```

•

```
__sleep()
```

— Method in class

```
DataTable
```

•

```
__toString()
```

— Method in class

```
DataTable
```

•

```
__construct()
```

— Method in class

```
BaseFilter
```

•

```
__construct()
```

— Method in class

```
AddColumnsProcessedMetrics
```

- 

`__construct()`

— *Method in class*

`AddColumnsProcessedMetricsGoal`

- 

`__construct()`

— *Method in class*

`AddSummaryRow`

- 

`__construct()`

— *Method in class*

`BeautifyRangeLabels`

- 

`__construct()`

— *Method in class*

`BeautifyTimeRangeLabels`

- 

`__construct()`

— *Method in class*

`CalculateEvolutionFilter`

- 

`__construct()`

— *Method in class*

`ColumnCallbackAddColumn`

- 

`__construct()`

— *Method in class*

`ColumnCallbackAddColumnQuotient`

- 

`__construct()`

— *Method in class*

`ColumnCallbackAddMetadata`

- 

`__construct()`

— *Method in class*

`ColumnCallbackDeleteRow`

-

`__construct()`

— *Method in class*

`ColumnDelete`

•

`__construct()`

— *Method in class*

`ExcludeLowPopulation`

•

`__construct()`

— *Method in class*

`GroupBy`

•

`__construct()`

— *Method in class*

`Limit`

•

`__construct()`

— *Method in class*

`MetadataCallbackAddMetadata`

- 

`__construct()`

— *Method in class*

`MetadataCallbackReplace`

- 

`__construct()`

— *Method in class*

`Pattern`

- 

`__construct()`

— *Method in class*

`PatternRecursive`

- 

`__construct()`

— *Method in class*

`ReplaceColumnNames`

- 

`__construct()`

— Method in class

ReplaceSummaryRowLabel

•

\_\_construct()

— Method in class

Sort

•

\_\_construct()

— Method in class

Truncate

•

\_\_toString()

— Method in class

Map

•

\_\_construct()

— Method in class

Row

- 

<code>__toString()</code>
---------------------------

— *Method in class*

Row
-----

- 

<code>__toString()</code>
---------------------------

— *Method in class*

Date
------

- 

<code>__construct()</code>
----------------------------

— *Method in class*

Mail
------

- 

<code>__construct()</code>
----------------------------

— *Method in class*

Notification
--------------

- 

<code>__toString()</code>
---------------------------

— *Method in class*

Period

- 

`__construct()`

— *Method in class*

Range

- 

`__construct()`

— *Method in class*

Plugin

- 

`__construct()`

— *Method in class*

Archiver

- 

`__construct()`

— *Method in class*

Controller

-

`__construct()`

— *Method in class*

`Settings`

•

`__construct()`

— *Method in class*

`ViewDataTable`

•

`__construct()`

— *Method in class*

`RankingQuery`

•

`__construct()`

— *Method in class*

`ScheduledTask`

•

`__construct()`

— *Method in class*

`Segment`

- 

```
__construct()
```

— *Method in class*

```
Setting
```

- 

```
__construct()
```

— *Method in class*

```
SystemSetting
```

- 

```
__construct()
```

— *Method in class*

```
UserSetting
```

- 

```
__construct()
```

— *Method in class*

```
Site
```

- 

```
__toString()
```

— Method in class

Site

•

\_\_construct()

— Method in class

View

•

\_\_set()

— Method in class

View

•

\_\_get()

— Method in class

View

•

\_\_construct()

— Method in class

Config

# PiwikTracker

---

PiwikTracker implements the Piwik Tracking Web API.

The PHP Tracking Client provides all features of the Javascript Tracker, such as Ecommerce Tracking, Custom Variable, Event tracking and more. Functions are named the same as the Javascript functions.

See introduction docs at: <http://piwik.org/docs/tracking-api/>

## Example: using the PHP PiwikTracker class

The following code snippet is an advanced example of how to track a Page View using the Tracking API PHP client.

```
$t = new PiwikTracker( $idSite = 1, 'http://example.org/piwik/' );
');

// Optional function calls

$t->setUserAgent( "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB) Firefox/3.6.6" );
$t->setBrowserLanguage( 'fr' );
$t->setLocalTime( '12:34:06' );
$t->setResolution( 1024, 768 );
$t->setBrowserHasCookies(true);
$t->setPlugins($flash = true, $java = true, $director = false);

// set a Custom Variable called 'Gender'
$t->setCustomVariable( 1, 'gender', 'male' );

// If you want to force the visitor IP, or force the server date time to a date in the past,
```

```

// it is required to authenticate the Tracking request by calling setTokenAuth

// You can pass the Super User token_auth or any user with 'admin' privilege on the website $idSite

$t->setTokenAuth( $token_auth );
$t->setIp( "134.10.22.1" );
$t->setForceVisitDateTime( '2011-04-05 23:55:02' );

// if you wanted to force to record the page view or conversion to a specific visitorId

// $t->setVisitorId( "33c31e01394bdc63" );

// Mandatory: set the URL being tracked

$t->setUrl( $url = 'http://example.org/store/list-category-toys/' );

// Finally, track the page view with a Custom Page Title

// In the standard JS API, the content of the <title> tag would be set as the page title

$t->doTrackPageView( 'This is the page title' );

```

## Example: tracking Ecommerce interactions

Here is an example showing how to track Ecommerce interactions on your website, using the PHP Tracking API. Usually, Ecommerce tracking is done using standard Javascript code, but it is very common to record Ecommerce interactions after the fact (for example, when payment is done with Paypal and user doesn't come back on the website after purchase). For more information about Ecommerce tracking in Piwik, check out the documentation: [Tracking Ecommerce in Piwik](#).

```

$t = new PiwikTracker( $idSite = 1, 'http://example.org/piwik/' );

// Force IP to the actual visitor IP
$t->setTokenAuth( $token_auth );
$t->setIp( "134.10.22.1" );

```

```

// Example 1: on a Product page, track an "Ecommerce Product view"

$t->setUrl( $url = 'http://www.mystore.com/Endurance-Shackletons-Legendary-Antarctic-Expedition' );

$t->setEcommerceView($sku = 'SKU0011', $name = 'Endurance - Shackleton', $category = 'Books');

$t->doTrackPageView( 'Endurance Shackletons Legendary Antarctic Expedition - Mystore.com' );


// Example 2: Tracking Ecommerce Cart containing 2 products

$t->addEcommerceItem($sku = 'SKU0011', $name = 'Endurance - Shackleton' , $category = 'Books', $price = 17, $quantity = 1);

// Note that when setting a product category, you can specify an array of up to 5 categories to track for this product

$t->addEcommerceItem($sku = 'SKU0321', $name = 'Am茅lie' , $categories = array('DVD Foreign','Best sellers','Our pick'), $price = 25, $quantity = 1);

$t->doTrackEcommerceCartUpdate($grandTotal = 42);


// Example 3: Tracking Ecommerce Order

$t->addEcommerceItem($sku = 'SKU0011', $name = 'Endurance - Shackleton' , $category = 'Books', $price = 17, $quantity = 1);

$t->addEcommerceItem($sku = 'SKU0321', $name = 'Am茅lie' , $categories = array('DVD Foreign','Best sellers','Our pick'), $price = 25, $quantity = 1);

$t->doTrackEcommerceOrder($orderId = 'B000111387', $grandTotal = 55.5, $subTotal = 42, $tax = 8, $shipping = 5.5, $discount = 10);

```

## Note: authenticating with the token\_auth

To set the visitor IP, or the date and time of the visit, or to force to record the visit (or page, or goal conversion) to a specific Visitor ID, you must call `setTokenAuth( $token_auth )`. The `token_auth` must be either the Super User `token_auth`, or the `token_auth` of any user with 'admin' permission for the website you are recording data against.

# Properties

This class defines the following properties:

- 

`$URL`

— Piwik base URL, for example `http://example.org/piwik/` Must be set before using the class by calling `PiwikTracker::$URL = 'http://yourwebsite.org/piwik/'`;

- 

`$DEBUG_APPEND_URL`

- 

`$DEBUG_LAST_REQUESTED_URL`

— Used in tests to output useful error messages.

`$URL`

Piwik base URL, for example `http://example.org/piwik/` Must be set before using the class by calling `PiwikTracker::$URL = 'http://yourwebsite.org/piwik/'`;

## Signature

- It is a

`string`

value.

`$DEBUG_APPEND_URL`

## Signature

- Its type is not specified.

`$DEBUG_LAST_REQUESTED_URL`

Used in tests to output useful error messages.

## Signature

- Its type is not specified.

## Methods

The class defines the following methods:

- 

`__construct()`

— Builds a PiwikTracker object, used to track visits, pages and Goal conversions for a specific website, by using the Piwik Tracking API.

- 

`setPageCharset()`

— By default, Piwik expects utf-8 encoded values, for example for the page URL parameter values, Page Title, etc.

- 

`setUrl()`

— Sets the current URL being tracked

- 

`setUrlReferrer()`

— Sets the URL referrer used to track Referrers details for new visits.

- 

`setGenerationTime()`

— Sets the time that generating the document on the server side took.

- 

`setUrlReferer()`

- 

```
setAttributionInfo()
```

— Sets the attribution information to the visit, so that subsequent Goal conversions are properly attributed to the right Referrer URL, timestamp, Campaign Name & Keyword.

- 

```
setCustomVariable()
```

— Sets Visit Custom Variable.

- 

```
getCustomVariable()
```

— Returns the currently assigned Custom Variable.

- 

```
setNewVisitorId()
```

— Sets the current visitor ID to a random new one.

- 

```
setIdSite()
```

— Sets the current site ID.

- 

```
setBrowserLanguage()
```

— Sets the Browser language.

- 

```
setUserAgent()
```

— Sets the user agent, used to detect OS and browser.

-

`setCountry()`

— Sets the country of the visitor.

•

`setRegion()`

— Sets the region of the visitor.

•

`setCity()`

— Sets the city of the visitor.

•

`setLatitude()`

— Sets the latitude of the visitor.

•

`setLongitude()`

— Sets the longitude of the visitor.

•

`enableBulkTracking()`

— Enables the bulk request feature.

•

`enableCookies()`

— Enable Cookie Creation - this will cause a first party VisitorId cookie to be set when the VisitorId is set or reset

•

`doTrackPageView()`

— Tracks a page view

- `doTrackEvent()`

  - Tracks an event
- `doTrackSiteSearch()`

  - Tracks an internal Site Search query, and optionally tracks the Search Category, and Search results Count.
- `doTrackGoal()`

  - Records a Goal conversion
- `doTrackAction()`

  - Tracks a download or outlink
- `addEcommerceItem()`

  - Adds an item in the Ecommerce order.
- `doTrackEcommerceCartUpdate()`

  - Tracks a Cart Update (add item, remove item, update item).
- `doBulkTrack()`

  - Sends all stored tracking actions at once.
- `doTrackEcommerceOrder()`

- Tracks an Ecommerce order.
- `setEcommerceView()`
  - Sets the current page view as an item (product) page view, or an Ecommerce Category page view.
- `getUrlTrackPageView()`
  - Builds URL to track a page view.
- `getUrlTrackEvent()`
  - Builds URL to track a custom event.
- `getUrlTrackSiteSearch()`
  - Builds URL to track a site search.
- `getUrlTrackGoal()`
  - Builds URL to track a goal with idGoal and revenue.
- `getUrlTrackAction()`
  - Builds URL to track a new action.
- `setForceVisitDateTime()`
  - Overrides server date and time for the tracking requests.

### `setIp()`

— Overrides IP address

- 

### `setVisitorId()`

— Forces the requests to be recorded for the specified Visitor ID rather than using the heuristics based on IP and other attributes.

- 

### `getVisitorId()`

— If the user initiating the request has the Piwik first party cookie, this function will try and return the ID parsed from this first party cookie (found in `$_COOKIE`).

- 

### `deleteCookies()`

— Deletes all first party cookies from the client

- 

### `getAttributionInfo()`

— Returns the currently assigned Attribution Information stored in a first party cookie.

- 

### `setTokenAuth()`

— Some Tracking API functionnality requires express authentication, using either the Super User token\_auth, or a user with 'admin' access to the website.

- 

### `setLocalTime()`

— Sets local visitor time

-

```
setResolution()
```

— Sets user resolution width and height.

•

```
setBrowserHasCookies()
```

ash; Sets if the browser supports cookies This is reported in "List of plugins" report in Piwik.

•

```
setDebugStringAppend()
```

— Will append a custom string at the end of the Tracking request.

•

```
setPlugins()
```

— Sets visitor browser supported plugins

•

```
disableCookieSupport()
```

— By default, PiwikTracker will read first party cookies from the request and write updated cookies in the response (using setrawcookie).

•

```
getRequestTimeout()
```

— Returns the maximum number of seconds the tracker will spend waiting for a response from Piwik.

•

```
setRequestTimeout()
```

— Sets the maximum number of seconds that the tracker will spend waiting for a response from Piwik.

```
__construct()
```

Builds a PiwikTracker object, used to track visits, pages and Goal conversions for a specific website, by using the Piwik Tracking API.

## Signature

- It accepts the following parameter(s):

- **\$idSite**

```
(
```

```
int
```

```
) —
```

Id site to be tracked

- **\$apiUrl**

```
(
```

```
string
```

```
) —
```

"http://example.org/piwik/" or "http://piwik.example.org/" If set, will overwrite PiwikTracker::\$URL

## setPageCharset()

By default, Piwik expects utf-8 encoded values, for example for the page URL parameter values, Page Title, etc.

It is recommended to only send UTF-8 data to Piwik. If required though, you can also specify another charset using this function.

## Signature

- It accepts the following parameter(s):

- **\$charset**

```
(
```

```
string
```

```
) —
```

- It does not return anything.

```
setUrl()
```

Sets the current URL being tracked

## Signature

- It accepts the following parameter(s):

```
• $url
```

```
(
```

```
string
```

```
) —
```

Raw URL (not URL encoded)

- It does not return anything.

```
setUrlReferrer()
```

Sets the URL referrer used to track Referrers details for new visits.

## Signature

- It accepts the following parameter(s):

```
• $url
```

```
(
```

```
string
```

```
) —
```

Raw URL (not URL encoded)

- It does not return anything.

```
setGenerationTime()
```

Sets the time that generating the document on the server side took.

## Signature

- It accepts the following parameter(s):

- **\$timeMs**

```
(
```

```
int
```

```
) —
```

Generation time in ms

- It does not return anything.

```
setUrlReferer()
```

## Signature

- It accepts the following parameter(s):

- **\$url**

- It does not return anything.

```
setAttributionInfo()
```

Sets the attribution information to the visit, so that subsequent Goal conversions are properly attributed to the right Referrer URL, timestamp, Campaign Name & Keyword.

This must be a JSON encoded string that would typically be fetched from the JS API: piwikTracker.getAttributionInfo() and that you have JSON encoded via JSON2.stringify()

If you call enableCookies() then these referral attribution values will be set to the 'ref' first party cookie storing referral information.

## See Also

- **function**

— getAttributionInfo() in <https://github.com/piwik/piwik/blob/master/js/piwik.js>

## Signature

- It accepts the following parameter(s):

- **\$jsonEncoded**

(

**string**

) —

JSON encoded array containing Attribution info

- It does not return anything.
- It throws one of the following exceptions:
  -

### **Exception**

## **setCustomVariable()**

Sets Visit Custom Variable.

See <http://piwik.org/docs/custom-variables/>

### Signature

- It accepts the following parameter(s):

- **\$id**

(

**int**

) —

Custom variable slot ID from 1-5

- **\$name**

(

**string**

) —

Custom variable name

- **\$value**

(

**string**

) —

Custom variable value

- **\$scope**

(

**string**

) —

Custom variable scope. Possible values: visit, page, event

- It does not return anything.
- It throws one of the following exceptions:

○

### **Exception**

**getCustomVariable()**

Returns the currently assigned Custom Variable.

If scope is 'visit', it will attempt to read the value set in the first party cookie created by Piwik Tracker (\$\_COOKIE array).

## See Also

- **Piwik.js**

— `getCustomVariable()`

## Signature

- It accepts the following parameter(s):

- **\$id**

(

int

) —

Custom Variable integer index to fetch from cookie. Should be a value from 1 to 5

- **\$scope**

(

string

) —

Custom variable scope. Possible values: visit, page, event

- *Returns:* (mixed

) —

An array with this format: array( 0 => CustomVariableName, 1 => CustomVariableValue ) or false

- It throws one of the following exceptions:

○

Exception

**setNewVisitorId()**

Sets the current visitor ID to a random new one.

## Signature

- It does not return anything.

**setIdSite()**

Sets the current site ID.

## Signature

- It accepts the following parameter(s):

- **\$idSite**

(

**int**

) —

- It does not return anything.

**setBrowserLanguage()**

Sets the Browser language.

Used to guess visitor countries when GeolP is not enabled

## Signature

- It accepts the following parameter(s):

- **\$acceptLanguage**

(

**string**

) —

For example "fr-fr"

- It does not return anything.

**setUserAgent()**

Sets the user agent, used to detect OS and browser.

If this function is not called, the User Agent will default to the current user agent.

## Signature

- It accepts the following parameter(s):

- **\$userAgent**

(

string

) —

- It does not return anything.

**setCountry()**

Sets the country of the visitor.

If not used, Piwik will try to find the country using either the visitor's IP address or language.

Allowed only for Admin/Super User, must be used along with setTokenAuth().

## Signature

- It accepts the following parameter(s):

• **\$country**

(

string

) —

- It does not return anything.

**setRegion()**

Sets the region of the visitor.

If not used, Piwik may try to find the region using the visitor's IP address (if configured to do so).

Allowed only for Admin/Super User, must be used along with setTokenAuth().

## Signature

- It accepts the following parameter(s):

• **\$region**

(

```
string
```

) —

- It does not return anything.

```
setCity()
```

Sets the city of the visitor.

If not used, Piwik may try to find the city using the visitor's IP address (if configured to do so).

Allowed only for Admin/Super User, must be used along with setTokenAuth().

## Signature

- It accepts the following parameter(s):

```
• $city
```

(

```
string
```

) —

- It does not return anything.

```
setLatitude()
```

Sets the latitude of the visitor.

If not used, Piwik may try to find the visitor's latitude using the visitor's IP address (if configured to do so).

Allowed only for Admin/Super User, must be used along with setTokenAuth().

## Signature

- It accepts the following parameter(s):

```
• $lat
```

(

```
float
```

) —

- It does not return anything.

### `setLongitude()`

Sets the longitude of the visitor.

If not used, Piwik may try to find the visitor's longitude using the visitor's IP address (if configured to do so).

Allowed only for Admin/Super User, must be used along with `setTokenAuth()`.

### Signature

- It accepts the following parameter(s):

- `$long`

(

`float`

) —

- It does not return anything.

### `enableBulkTracking()`

Enables the bulk request feature.

When used, each tracking action is stored until the `doBulkTrack` method is called. This method will send all tracking data at once.

### Signature

- It does not return anything.

### `enableCookies()`

Enable Cookie Creation - this will cause a first party VisitorId cookie to be set when the VisitorId is set or reset

### Signature

- It accepts the following parameter(s):

- `$domain`

(

string

) —

(optional) Set first-party cookie domain. Accepted values: example.com, \*.example.com (same as .example.com) or subdomain.example.com

- **\$path**

(

string

) —

(optional) Set first-party cookie path

- It does not return anything.

**doTrackPageView()**

Tracks a page view

## Signature

- It accepts the following parameter(s):

- **\$documentTitle**

(

string

) —

Page title as it will appear in the Actions > Page titles report

- *Returns:* (

mixed

) —

Response string or true if using bulk requests.

**doTrackEvent()**

Tracks an event

## Signature

- It accepts the following parameter(s):

- **\$category**

(

**string**

) —

The Event Category (Videos, Music, Games...)

- **\$action**

(

**string**

) —

The Event's Action (Play, Pause, Duration, Add Playlist, Downloaded, Clicked...)

- **\$name**

(

**string**

) —

(optional) The Event's object Name (a particular Movie name, or Song name, or File name...)

- **\$value**

(

**float**

) —

(optional) The Event's value

- *Returns:* (

```
mixed
```

```
) —
```

Response string or true if using bulk requests.

### doTrackSiteSearch()

Tracks an internal Site Search query, and optionally tracks the Search Category, and Search results Count.

These are used to populate reports in Actions > Site Search.

#### Signature

- It accepts the following parameter(s):

- **\$keyword**

```
(
```

```
string
```

```
) —
```

Searched query on the site

- **\$category**

```
(
```

```
string
```

```
) —
```

(optional) Search engine category if applicable

- **\$countResults**

```
(
```

```
bool
```

```
|
```

```
int
```

) —

(optional) results displayed on the search result page. Used to track "zero result" keywords.

- *Returns:* (

`mixed`

) —

Response or true if using bulk requests.

`doTrackGoal()`

Records a Goal conversion

## Signature

- It accepts the following parameter(s):

- `$idGoal`

(

`int`

) —

Id Goal to record a conversion

- `$revenue`

(

`float`

) —

Revenue for this conversion

- *Returns:* (

`mixed`

) —

Response or true if using bulk request

`doTrackAction()`

Tracks a download or outlink

## Signature

- It accepts the following parameter(s):

- **\$actionUrl**

(

**string**

) —

URL of the download or outlink

- **\$actionType**

(

**string**

) —

Type of the action: 'download' or 'link'

- *Returns:* (

**mixed**

) —

Response or true if using bulk request

**addEcommerceItem()**

Adds an item in the Ecommerce order.

This should be called before doTrackEcommerceOrder(), or before doTrackEcommerceCartUpdate(). This function can be called for all individual products in the cart (or order). SKU parameter is mandatory. Other parameters are optional (set to false if value not known). Ecommerce items added via this function are automatically cleared when doTrackEcommerceOrder() or getUrlTrackEcommerceOrder() is called.

## Signature

- It accepts the following parameter(s):

- **\$sku**

(

**string**

)—

(required) SKU, Product identifier

- **\$name**

(

**string**

)—

(optional) Product name

- **\$category**

(

**string**

|

**array**

)—

(optional) Product category, or array of product categories (up to 5 categories can be specified for a given product)

- **\$price**

(

**float**

|

**int**

)—

(optional) Individual product price (supports integer and decimal prices)

- **\$quantity**

(

**int**

) —

(optional) Product quantity. If not specified, will default to 1 in the Reports

- It does not return anything.
- It throws one of the following exceptions:

○

### **Exception**

## **doTrackEcommerceCartUpdate()**

Tracks a Cart Update (add item, remove item, update item).

On every Cart update, you must call addEcommerceItem() for each item (product) in the cart, including the items that haven't been updated since the last cart update. Items which were in the previous cart and are not sent in later Cart updates will be deleted from the cart (in the database).

## Signature

- It accepts the following parameter(s):

- **\$grandTotal**

(

**float**

) —

Cart grandTotal (typically the sum of all items' prices)

- *Returns:* (  
**mixed**)

) —

Response or true if using bulk request

### doBulkTrack()

Sends all stored tracking actions at once.

Only has an effect if bulk tracking is enabled.

To enable bulk tracking, call `enableBulkTracking()`.

### Signature

- *Returns:* (

`string`

) —

Response

- It throws one of the following exceptions:

○

### Exception

### doTrackEcommerceOrder()

Tracks an Ecommerce order.

If the Ecommerce order contains items (products), you must call first the `addEcommerceItem()` for each item in the order. All revenues (grandTotal, subTotal, tax, shipping, discount) will be individually summed and reported in Piwik reports. Only the parameters `$orderId` and `$grandTotal` are required.

### Signature

- It accepts the following parameter(s):

- `$orderId`

(

`string`

|

`int`

) —

(required) Unique Order ID. This will be used to count this order only once in the event the order page is reloaded several times. orderId must be unique for each transaction, even on different days, or the transaction will not be recorded by Piwik.

- `$grandTotal`

(

`float`

) —

(required) Grand Total revenue of the transaction (including tax, shipping, etc.)

- `$subTotal`

(

`float`

) —

(optional) Sub total amount, typically the sum of items prices for all items in this order (before Tax and Shipping costs are applied)

- `$tax`

(

`float`

) —

(optional) Tax amount for this order

- `$shipping`

(

`float`

) —

(optional) Shipping amount for this order

- `$discount`

(

`float`

) —

(optional) Discounted amount in this order

- *Returns:* (

`mixed`

) —

Response or true if using bulk request

### `setEcommerceView()`

Sets the current page view as an item (product) page view, or an Ecommerce Category page view.

This must be called before doTrackPageView() on this product/category page. It will set 3 custom variables of scope "page" with the SKU, Name and Category for this page view.

Note: Custom Variables of scope "page" slots 3, 4 and 5 will be used.

On a category page, you may set the parameter \$category only and set the other parameters to false.

Tracking Product/Category page views will allow Piwik to report on Product & Categories conversion rates (Conversion rate = Ecommerce orders containing this product or category / Visits to the product or category)

### Signature

- It accepts the following parameter(s):

- `$sku`

(

`string`

) —

Product SKU being viewed

- **\$name**

(

**string**

)—

Product Name being viewed

- **\$category**

(

**string**

|

**array**

)—

Category being viewed. On a Product page, this is the product's category. You can also specify an array of up to 5 categories for a given page view.

- **\$price**

(

**float**

)—

Specify the price at which the item was displayed

- It does not return anything.

### **getUrlTrackPageView()**

Builds URL to track a page view.

### See Also

- **doTrackPageView()**

## Signature

- It accepts the following parameter(s):

- **\$documentTitle**

(

**string**

) —

Page view name as it will appear in Piwik reports

- *Returns:* (

**string**

) —

URL to piwik.php with all parameters set to track the pageview

**getUrlTrackEvent()**

Builds URL to track a custom event.

## See Also

- **doTrackEvent()**

## Signature

- It accepts the following parameter(s):

- **\$category**

(

**string**

) —

The Event Category (Videos, Music, Games...)

- **\$action**

(

```
string
```

) —

The Event's Action (Play, Pause, Duration, Add Playlist, Downloaded, Clicked...)

- **\$name**

(

```
string
```

) —

(optional) The Event's object Name (a particular Movie name, or Song name, or File name...)

- **\$value**

(

```
float
```

) —

(optional) The Event's value

- *Returns:* (

```
string
```

) —

URL to piwik.php with all parameters set to track the pageview

```
getUrlTrackSiteSearch()
```

Builds URL to track a site search.

## See Also

- [doTrackSiteSearch\(\)](#)

## Signature

- It accepts the following parameter(s):

- **\$keyword**

(

string

)—

- **\$category**

(

string

)—

- **\$countResults**

(

int

)—

- It returns a

string

value.

### getUrlTrackGoal()

Builds URL to track a goal with idGoal and revenue.

## See Also

- [doTrackGoal\(\)](#)

## Signature

- It accepts the following parameter(s):

- **\$idGoal**

(

int

)—

Id Goal to record a conversion

- `$revenue`

(

`float`

) —

Revenue for this conversion

- *Returns:* (  
`string`

) —

URL to piwik.php with all parameters set to track the goal conversion

`getUrlTrackAction()`

Builds URL to track a new action.

## See Also

- `doTrackAction()`

## Signature

- It accepts the following parameter(s):

- `$actionUrl`

(

`string`

) —

URL of the download or outlink

- `$actionType`

(

`string`

) —

Type of the action: 'download' or 'link'

- *Returns:* (  
`string`)

) —

URL to piwik.php with all parameters set to track an action

### `setForceVisitDateTime()`

Overrides server date and time for the tracking requests.

By default Piwik will track requests for the "current datetime" but this function allows you to track visits in the past. All times are in UTC.

Allowed only for Super User, must be used along with `setTokenAuth()`

## See Also

- `setTokenAuth()`

## Signature

- It accepts the following parameter(s):

- `$dateTime`

(

`string`

) —

Date with the format 'Y-m-d H:i:s', or a UNIX timestamp

- It does not return anything.

### `setIp()`

Overrides IP address

Allowed only for Super User, must be used along with `setTokenAuth()`

## See Also

- `setTokenAuth()`

## Signature

- It accepts the following parameter(s):

- `$ip`

(

`string`

) —

IP string, eg. 130.54.2.1

- It does not return anything.

`setVisitorId()`

Forces the requests to be recorded for the specified Visitor ID rather than using the heuristics based on IP and other attributes.

Allowed only for Admin/Super User, must be used along with `setTokenAuth()`.

You may set the Visitor ID based on a user attribute, for example the user email:

```
$v->setVisitorId( substr(md5( $userEmail ), 0, 16));
```

If not set, the visitor ID will be fetched from the 1st party cookie, or will be set to a random UUID.

## See Also

- `setTokenAuth()`

## Signature

- It accepts the following parameter(s):

- `$visitorId`

(

`string`

) —

16 hexadecimal characters visitor ID, eg. "33c31e01394bdc63"

- It does not return anything.
- It throws one of the following exceptions:
  -

### Exception

#### getVisitorId()

If the user initiating the request has the Piwik first party cookie, this function will try and return the ID parsed from this first party cookie (found in `$_COOKIE`).

If you call this function from a server, where the call is triggered by a cron or script not initiated by the actual visitor being tracked, then it will return the random Visitor ID that was assigned to this visit object.

This can be used if you wish to record more visits, actions or goals for this visitor ID later on.

### Signature

- *Returns:* (  
    string

) —

16 hex chars visitor ID string

#### deleteCookies()

Deletes all first party cookies from the client

### Signature

- It does not return anything.

#### getAttributionInfo()

Returns the currently assigned Attribution Information stored in a first party cookie.

This function will only work if the user is initiating the current request, and his cookies can be read by PHP from the `$_COOKIE` array.

### See Also

- [Piwik.js](#)

— [getAttributionInfo\(\)](#)

## Signature

- *Returns:* (  
    **string**

) —

JSON Encoded string containing the Referrer information for Goal conversion attribution. Will return false if the cookie could not be found

## **setTokenAuth()**

Some Tracking API functionnality requires express authentication, using either the Super User token\_auth, or a user with 'admin' access to the website.

The following features require access: - force the visitor IP - force the date & time of the tracking requests rather than track for the current datetime - force Piwik to track the requests to a specific VisitorId rather than use the standard visitor matching heuristic

## Signature

- It accepts the following parameter(s):

- **\$token\_auth**

(

**string**

) —

token\_auth 32 chars token\_auth string

- It does not return anything.

## **setLocalTime()**

Sets local visitor time

## Signature

- It accepts the following parameter(s):

- **\$time**

(

```
string
```

) —

HH:MM:SS format

- It does not return anything.

```
setResolution()
```

Sets user resolution width and height.

## Signature

- It accepts the following parameter(s):

```
• $width
```

(

```
int
```

) —

```
• $height
```

(

```
int
```

) —

- It does not return anything.

```
setBrowserHasCookies()
```

Sets if the browser supports cookies This is reported in "List of plugins" report in Piwik.

## Signature

- It accepts the following parameter(s):

```
• $bool
```

(

```
bool
```

) —

- It does not return anything.

**setDebugStringAppend()**

Will append a custom string at the end of the Tracking request.

## Signature

- It accepts the following parameter(s):

• **\$string**

(

**string**

) —

- It does not return anything.

**setPlugins()**

Sets visitor browser supported plugins

## Signature

- It accepts the following parameter(s):

• **\$flash**

(

**bool**

) —

• **\$java**

(

**bool**

) —

• **\$director**

(

bool

)—

- \$quickTime

(

bool

)—

- \$realPlayer

(

bool

)—

- \$pdf

(

bool

)—

- \$windowsMedia

(

bool

)—

- \$gears

(

bool

)—

- **\$silverlight**

(

bool

)—

- It does not return anything.

### **disableCookieSupport()**

By default, PiwikTracker will read first party cookies from the request and write updated cookies in the response (using setrawcookie).

This can be disabled by calling this function.

#### Signature

- It does not return anything.

### **getRequestTimeout()**

Returns the maximum number of seconds the tracker will spend waiting for a response from Piwik.

Defaults to 600 seconds.

#### Signature

- It does not return anything.

### **setRequestTimeout()**

Sets the maximum number of seconds that the tracker will spend waiting for a response from Piwik.

#### Signature

- It accepts the following parameter(s):

- **\$timeout**

(

int

)—

- It does not return anything.
- It throws one of the following exceptions:
  -

### Exception

# Javascript Tracking client How-to

Piwik is equipped with a powerful JavaScript Tracking API. Advanced users can use the Piwik tracking code to customize the way some of the web analytics data is recorded in Piwik.

## Where can I Find the Piwik Tracking Code?

To use all the features described in this page, you need to use the latest version of the tracking code. To find the tracking code for your website, please follow the steps below:

- Log in to Piwik with your admin or Super User account
- Click on Settings to access the administration area
- Click on Websites to list the websites that you are tracking in Piwik
- Click on View Tracking code for the website you wish to track
- You can now copy and paste the JavaScript Tracking code into your pages, just before the </body> tag.

The Piwik Tracking code looks as follows:

```
<!-- Piwik -->

<script type="text/javascript">

var _paq = _paq || [];
(function(){ var u=(("https:" == document.location.protocol) ? "https://{$PIWIK_URL}/" : "http://{$PIWIK_URL}/");
_paq.push(['setSiteId', {$IDSITE}]);
```

```

_paq.push(['setTrackerUrl', u+'piwik.php']);

_paq.push(['trackPageView']);

_paq.push(['enableLinkTracking']);

var d=document, g=d.createElement('script'), s=d.getElementsByTagName('script')[0]; g.type='text/javascript'; g.defer=true; g.async=true; g.src=u+'piwik.js';

s.parentNode.insertBefore(g,s); })();

</script>

```

<!-- End Piwik Code -->

In your Piwik tracking code, {\$PIWIK\_URL} would be replaced by your Piwik URL and {\$IDSITE} would be replaced by the idsite of the website you are tracking in Piwik.

This code might look a bit strange to those of you familiar with JavaScript, but that is because it is made to run asynchronously. In other words, browsers will not wait for the piwik.js file to be downloaded in order to show your page.

For asynchronous tracking, configuration and tracking calls are pushed onto the global

```

_paq
array for execution, independent of the asynchronous loading of piwik.js. The format is:
_paq.push([ 'API_method_name', parameter_list ]);
```

You can also push functions to be executed. For example:

```

var visitor_id;

_paq.push([ function() { visitor_id = this.getVisitorId(); }]);
```

or for example, to fetch a custom variable (name,value) using the asynchronous code:

```

_paq.push(['setCustomVariable','1','VisitorType','Member']);
_paq.push([ function() { var customVariable = this.getCustomVariable(1); }]);
```

You can push to the \_paq array even after the piwik.js file has been loaded and run.

If your Piwik tracking code doesn't look like this one, you may be using the deprecated version. Older versions still work as expected and will track your visitors, but we highly recommend that you update your pages to use the most recent tracking code.

# Features of the JavaScript Tracker

## Customize the Page Name Displayed in Piwik

By default, Piwik uses the URL of the current page as the page title in the Piwik interface. If your URLs are not simple, or if you want to customize the way Piwik tracks your pages, you can specify the page title to use in the JavaScript code.

A common use is to set the HTML Title value as the document title:

```
[...]
```

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

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

```
[...]
```

If you track **multiple sub-domains in the same website**, you may want your page titles to be prefixed by the sub-domain make it easy for you to see the traffic and data for each sub-domain. You can do so simply in JavaScript:

```
[...]
```

```
_paq.push(['setDocumentTitle', document.domain + "/" + document.title]);
```

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

```
[...]
```

Advanced users can also dynamically generate the page name, for example, in PHP:

```
[...]
```

```
_paq.push(['setDocumentTitle', "<?php echo $myPageTitle ?>"]);  
  
_paq.push(['trackPageView']);  
  
[...]
```

## Manually Trigger a Page View on Click or on JS Event

By default, Piwik tracks page views when the Javascript tracking code loads and executes on each page view. However, on modern websites or web applications, user interactions do not necessarily involve loading a new page. For example, when users click on a JavaScript link, or when they click on a tab (which triggers a JS event), or when they interact with elements of the user interface, you can still track these interactions with Piwik.

To track any user interaction or click with Piwik, you can manually call the Javascript function

```
trackPageView()
```

. For example, if you wanted to track a click on a JavaScript menu, you could write:

```
[...]
```

```
<a href="#" onclick="javascript:_paq.push(['trackPageView', 'Menu/Freedom']);">Freedom page</a>
```

```
[...]
```

## Manually Trigger a Conversion for a Goal

By default, Goals in Piwik are defined as "matching" parts of the URL (starts with, contains, or regular expression matching). You can also track goals for given page views, downloads, or outlink clicks.

In some situations, you may want to register conversions on other types of actions, for example:

- when a user submits a form
- when a user has stayed more than a given amount of time on the page
- when a user does some interaction in your Flash application
- when a user has submitted his cart and has done the payment: you can give the Piwik tracking code to the payment website which will then register the conversions in your Piwik database, with the conversion's custom revenue

To trigger a goal using the Piwik JavaScript Tracking, you can simply do:

```
[...]
```

```
// logs a conversion for goal 1  
  
_paq.push(['trackGoal', 1]);
```

```
[...]
```

You can also register a conversion for this goal with a custom revenue. For example, you can generate the call to trackGoal dynamically to set the revenue of the transaction:

```
[...]
```

```
/* Logs a conversion for goal 1 with the custom revenue set */  
  
_paq.push(['trackGoal', 1, <?php echo $cart->getCartValue();  
?>]);
```

```
[...]
```

Find more information about goal tracking in Piwik in the [Tracking Goals](#) documentation.

# Tracking Ecommerce Orders, Cart Updates and Product/Category Page Views

Piwik allows for advanced and powerful Ecommerce tracking. Check out the [Ecommerce Analytics](#) documentation for more information about Ecommerce reports and how to set up Ecommerce tracking.

## Tracking internal Search Keywords, categories, and No Result Search Keywords

Piwik offers advanced [Site Search Analytics](#) feature, letting you track how your visitors use your internal website search engine. By default, Piwik can read URL parameters that will contain the search keyword. However, you can also record the site search keyword manually using the Javascript function

```
trackSiteSearch(...)
```

In your website, in standard pages, you would typically have a call to record Page views via

```
piwikTracker.trackPageView()
```

. On your search result page, you would call instead

```
piwikTracker.trackSiteSearch(keyword, category, searchCount)
```

function to record the internal search request. Note: the 'keyword' parameter is required, but category and searchCount are optional.

```
[...]
```

```
_paq.push(['trackSiteSearch',
```

```
    // Search keyword searched for
```

```
    "Banana",
```

```
    // Search category selected in your search engine. If you do  
    // not need this, set to false
```

```
    "Organic Food",
```

```
    // Number of results on the Search results page. Zero indicates  
    // a 'No Result Search Keyword'. Set to false if you don't know
```

```
    0
```

```
]);
```

```
// We recommend not to call trackPageView() on the Site Search  
Result page  
// _paq.push(['trackPageView']);  
[...]
```

We also highly recommend to set the searchCount parameter, as Piwik will specifically report "No Result Keywords", ie. Keywords that were searched, but did not return any result. It is usually very interesting to know what users search for but can't find (yet?) on your website. Learn more about [Site Search Analytics in the User Doc](#).

## Custom Variables

Custom variables are a powerful feature that enable you to track custom values for each visit, and/or each page view. Please see the [Tracking custom variables](#) documentation page for general information.

You can set up to 5 custom variables (name and value) for each visit to your website, and/or up to 5 custom variables for each page view. If you set a custom variable to a visitor, when he comes back 1 hour or 2 days later, it will be a new visit and his/her custom variables will be empty.

There are two "scopes" which you can set your custom variables to. The "scope" is the 4th parameter of the function

```
setCustomVariable()
```

- when scope = "visit", the custom variable's name and value will be stored in the visit in the database. You can therefore store up to 5 custom variables of scope "visit" for each visit.
- when scope = "page", the custom variable's name and value will be stored for the page view being tracked. You can therefore store up to 5 custom variables of scope "page" for each page view.

Custom variable statistics are reported in Piwik under **Visitors > custom variables**. Both custom variables of scope "visit" and "page" are aggregated in this report.

### Set a Custom Variable for the Visit

```
setCustomVariable (index, name, value, scope = "visit")
```

This function is used to create, or update a custom variable name and value. For example, imagine you want to store in each visit the gender of the user. You would store the custom variable with a name = "gender", value = "male" or "female".

**Important:** a given custom variable name must always be stored in the same "index". 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 the new custom variable stored in index 1.

```
[...]  
_paq.push(['setCustomVariable',  
    // Index, the number from 1 to 5 where this custom variable  
    // name is stored  
    1,  
    // Name, the name of the variable, for example: Gender, VisitorType  
    "Gender",  
    // Value, for example: "Male", "Female" or "new", "engaged",  
    // "customer"  
    "Male",  
    // Scope of the custom variable, "visit" means the custom variable applies to the current visit  
    "visit"  
]);  
  
_paq.push(['trackPageView']);  
[...]
```

You only need to set a variable with scope "visit" once, and the value will be recorded for the whole visit.

## Set a Custom Variable for a Page View

```
setCustomVariable (index, name, value, scope = "page")
```

As well as tracking custom variables for "visits", it is sometimes useful to track custom variables for each page view separately. For example, for a "News" website or blog, a given article may be categorized into one or several categories. In this case, you could set one or several custom variables with

```
name="category"  
, one with  
value="Sports"  
and another with  
value="Europe"
```

if the article is classified in Sports and Europe Categories. The custom variables report will then report on how many visits and page views were in each of your website's categories. This information can be difficult to obtain with standard Piwik reports because they report on "Best Page URLs" and "Best Page Titles" which might not contain the "category" information.

```
[...]  
// Track 2 custom variables with the same name, but in different slots.  
// You will then access the statistics about your articles' categories in the 'Visitors > custom variables' report  
_paq.push(['setCustomVariable', 1, 'Category', 'Sports', 'page']);  
  
// Track the same name but in a different Index  
_paq.push(['setCustomVariable', 2, 'Category', 'Europe', 'page']);  
// Here you could track other custom variables with scope "page" in Index 3, 4 or 5  
// The order is important: first setCustomVariable is called and then trackPageView records the request  
  
_paq.push(['trackPageView']);  
[...]
```

**Important:** It is possible to store a custom variables of scope "visit" in "index" 1, and store a different custom variable of scope "page" in the same "index" 1. Therefore, you can technically **track up to 10 custom variables names and values on every page** of your website (5 with a "page" scope stored in the actual page view, 5 with a "visit" scope stored in the visit).

```
[...]  
_paq.push(['setCustomVariable',  
// Index, the number from 1 to 5 where this custom variable name is stored for the current page view  
 1,  
// Name, the name of the variable, for example: Category, Sub-category, UserType  
]);
```

```
"category",
    // Value, for example: "Sports", "News", "World", "Business",
    // etc.
    "Sports",
    // Scope of the custom variable, "page" means the custom variable applies to the current page view
    "page"
]);

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

### Delete a Custom Variable

```
deleteCustomVariable (index, scope )
```

If you created a custom variable and then decide to remove this variable from a visit or page view, you can use deleteCustomVariable.

To persist the change in the Piwik server, you must call the function before the call to

```
trackPageView();
[...]
```

```
_paq.push(['deleteCustomVariable', 1, "visit"]); // Delete the variable in index 1 stored for the current visit
```

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

```
[...]
```

### Get the Name and Value of a Custom Variable

```
getCustomVariable (index, scope )
```

This function is mostly useful if scope = "visit".

In this case, custom variables are recorded in a first party cookie for the duration of the visit (30 minutes after the last action). You can actually retrieve the custom variable

names and values using `piwikTracker.getCustomVariable`. If there is no custom variable in the requested index, it will return false.

```
[...]
```

```
_paq.push([ function() {  
  
    var customVariable = this.getCustomVariable( 1, "visit" );  
    // Returns the custom variable: [ "gender", "male" ]  
  
    // do something with customVariable...  
  
}]);  
  
  
_paq.push(['trackPageView']);  
  
[...]
```

## Cookie Configuration for Domains and Subdomains

Piwik uses first-party cookies to keep track some information (number of visits, original referrer, and unique visitor ID). First-party cookies ensure higher user privacy (since cookies are not sent to a third-party server), and are therefore accepted in most browsers by default.

Piwik creates a set of cookies for each domain and subdomain. If you want to track some subdomains and share the same cookie for accurate statistics, it is necessary to customize the Piwik Tracking code. Check out the examples for the various configurations.

## If you Track Only One Domain Name or Subdomain for a Single Website in Piwik

This is the standard use case. Piwik tracks the visits of one domain name with no subdomain, in a single Piwik website.

```
[...]  
// Default Tracking code  
  
_paq.push(['setSiteId', 1]);  
  
_paq.push(['setTrackerUrl', u+'piwik.php']);  
  
_paq.push(['trackPageView']);  
  
[...]
```

## If you Track One Domain Name and Several Subdomains for a Single Website in Piwik

If you want to record visits for the main domain name as well as its subdomains, you would want to share cookies across all domains. You can do so by calling

```
setCookieDomain()  
, in all subdomains tracking codes.  
[...]  
  
_paq.push(['setSiteId', 1]);  
  
_paq.push(['setTrackerUrl', u+'piwik.php']);
```

```
// Same cookie as: example.com, www.example.com, subdomain.example.com, ...
_paq.push(['setCookieDomain', '*.example.com']);

// Set cookie domain for subdomains
_paq.push(['setDomains', '*.example.com']); // Download & Click tracking alias domains

_paq.push(['trackPageView']);

[...]
```

## If you Track Domain Subdirectories or Pages in Different Websites in Piwik

By default, Piwik uses only one cookie for a domain name, and all its pages and subdirectories.

There may be cases where you track a subdirectory as a separate website in Piwik. If a visitor visits more than a few subdirectories, this will cause some inaccuracy in your reports: time on site, number of visits, conversion referrer, returning and new visitors. In this use case, you can ensure your reports stay accurate by creating a different cookie for each subpath you are tracking in different Piwik websites. The function

`setCookiePath()`

is used to set the Cookie path.

For example, if your website has user profiles, you could track each user profile page analytics as a unique website in Piwik. In the main domain homepage, you would use the default tracking code.

[...]

```
// idSite = X for the Homepage
_paq.push(['setSiteId', X]);
```

```
_paq.push(['setTrackerUrl', u+'piwik.php']);
```

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

```
[...]
```

In the /user/MyUsername page, you would write:

```
[...]
```

```
// The idSite Y will be different from other user pages
```

```
_paq.push(['setSiteId', Y]);
```

```
_paq.push(['setTrackerUrl', u+'piwik.php']);
```

```
_paq.push(['setCookiePath', '/user/MyUsername']);
```

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

```
[...]
```

For more information about tracking websites and subdomains in Piwik, see the FAQ: [How to configure Piwik to monitor several websites, domains and sub-domains](#)

## Ignore Specific Domains or Subdomains in the "Outlink" Click Tracking

By default all links to domains other than the current domain have click tracking enabled, and each click will be counted as an outlink. If you use multiple domains and subdomains, you may see clicks on your subdomains appearing in the Pages > Outlinks report.

If you only want clicks to external websites to appear in your outlinks report, you can use the function

```
setDomains()
```

to specify the list of alias domains or subdomains. Wildcard domains (\*.example.org) are supported to let you easily ignore clicks to all subdomains.

```
[...]
```

```
// Don't track Outlinks on all clicks pointing to *.hostname1.com or *.hostname2.com
```

```
// Note: the currently tracked website is added to this array automatically
```

```
_paq([ 'setDomains', ["*.hostname1.com", "hostname2.com"]]);
```

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

```
[...]
```

## Disable the Download & Outlink Tracking

By default, the Piwik tracking code enables clicks and download tracking. To disable all automatic download and outlink tracking, you must remove the call to the

```
enableLinkTracking()
```

function:

```
[...]
```

```
// we comment out the function that enables link tracking
```

```
// _paq.push([ 'enableLinkTracking']);
```

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

```
[...]
```

## Disable the Download & Outlink Tracking for Specific CSS Classes

You can disable automatic download and outlink tracking for links with specific CSS classes:

```
[...]
```

```
// you can also pass an array of strings
```

```
_paq.push(['setIgnoreClasses', "no-tracking"]);
```

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

```
[...]
```

This will result in clicks on a link

```
<a href='http://example.com' class='no-tracking'>Test</a>
```

not being counted.

## Disable Download or Outlink Tracking on a Specific Link

If you want to always ignore download or outlink tracking on a specific link, you can add the 'piwik\_ignore' css class to it:

```
<a href='http://builds.piwik.org/latest.zip' class='piwik_ignore'>File I don't want to track as a download</a>
```

# Force a Click on a Link to be Recorded as a Download in Piwik

If you want Piwik to consider a given link as a download, you can add the 'piwik\_download' css class to the link:

```
<a href='last.php' class='piwik_download'>Link I want to track  
as a download</a>
```

Note: you can customize and rename the CSS class used to force a click to be recorded as a download:

```
[...]
```

```
// now all clicks on links with the css class "downLoad" will be  
counted as downLoads
```

```
// you can also pass an array of strings
```

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

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

```
[...]
```

# Force a Click on a Link to be Recorded as an Outlink

If you want Piwik to consider a given link as an outlink (links to the current domain or to one of the alias domains), you can add the 'piwik\_link' css class to the link:

```
<a href='http://mysite.com/partner/' class='piwik_link'>Link I  
want to track as an outlink</a>
```

Note: you can customize and rename the CSS class used to force a click to being recorded as an outlink:

```
[...]
```

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

Alternatively, you can use JavaScript to manually trigger a click on an outlink (it will work the same for page views or file downloads). In this example, custom outlink is triggered when the email address is clicked:

```
<a href="mailto:namexyz@mydomain.co.uk" target="_blank" onClick  
="javascript:_paq.push(['trackLink', 'http://mydomain.co.uk/mai  
lto/Agent namexyz', 'link']);">namexyz@mydomain.co.uk </a>
```

## Changing the Pause Timer

When a user clicks to download a file, or clicks on an outbound link, Piwik records it. In order to do so, it adds a small delay before the user is redirected to the requested file or link. The default value is 500ms, but you can set it to a shorter length of time. It should be noted, however, that doing so results in the risk that this period of time is not long enough for the data to be recorded in Piwik.

```
[...]
```

```
_paq.push(['setLinkTrackingTimer', 250]); // 250 milliseconds  
  
_paq.push(['trackPageView']);
```

```
[...]
```

## Changing the List of File Extensions to Track as "Downloads"

By default, any file ending with one of these extensions will be considered a 'download' in the Piwik interface:

```
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|
mpeg|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
```

To replace the list of extensions you want to track as file downloads, you can use

```
setDownloadExtensions( string )
:
[...]

// we now only track clicks on images
_paq.push(['setDownloadExtensions', "jpg|png|gif"]);

_paq.push(['trackPageView']);

[...]
```

If you want to track a new filetype, you can just add it to the list by using

```
addDownloadExtensions( filetype )
:
[...]

// clicks on URLs finishing by mp5 or mp6 will be counted as downloads
```

```
_paq.push(['addDownloadExtensions', "mp5|mp6"]);  
  
_paq.push(['trackPageView']);  
  
[...]
```

## Multiple Piwik Trackers

It is possible to track a page using multiple Piwik trackers that point to the same or different Piwik servers. To improve page loading time, you can load piwik.js once. Each call to

```
Piwik.getTracker()
```

returns a unique Piwik Tracker object (instance) which can be configured.

Note: When using multiple trackers, you must use the synchronous Javascript tracker object (ie. this will not work with the asynchronous tag):

```
<script type="text/javascript">  
  
try {  
  
    var piwikTracker = Piwik.getTracker("http://URL_1/piwik.php", 1);  
  
    piwikTracker.trackPageView();  
  
    var piwik2 = Piwik.getTracker("http://URL_2/piwik.php", 4);  
  
    piwik2.trackPageView();  
  
} catch( err ) {}  
  
</script>
```

Note that you can also set the website ID and the Piwik tracker URL manually, instead of setting them in the getTracker call:

```
// we replace Piwik.getTracker("http://example.com/piwik/", 12)
```

```
var piwikTracker = Piwik.getTracker();  
  
piwikTracker.setSiteId( 12 );  
  
piwikTracker.setTrackerUrl( "http://example.com/piwik/" );  
  
piwikTracker.trackPageView();
```

## List of all Methods Available in the Tracking API

*Requesting the Tracker Instance from the Piwik Class*

- **Piwik.getTracker( trackerUrl, siteId )**

- get a new instance of the Tracker

- [Google Analytics equivalent] \_getTracker(account)
- [Yahoo! Analytics equivalent] getTracker(account)

- **Piwik.getAsyncTracker()**

- get the internal instance of the Tracker used for asynchronous tracking

*Using the Tracker Object*

- **enableLinkTracking( enable )**

- Install link tracking on all applicable link elements. Set the enable parameter to true to use a pseudo-click handler to track browsers (such as Firefox) which don't generate click events for the middle mouse button. By default only "true" mouse click events are handled.

- **addListener( element )**

- Add click listener to a specific link element. When clicked, Piwik will log the click automatically.

- **setRequestMethod( method )**

- Set the request method to either "GET" or "POST". (The default is "GET".) To use the POST request method, the Piwik host must be the same as the tracked website host (Piwik installed in the same domain as your tracked website).

- **trackGoal( idGoal, [customRevenue] );**

- Manually log a conversion for the goal idGoal, passing in the custom revenue customRevenue if specified

- **trackLink( url, linkType )**

- Manually log a click from your own code. url is the full URL which is to be tracked as a click. linkType can either be 'link' for an outlink or 'download' for a download.

- **trackPageView([customTitle])**

- Log visit to this page

- [Google Analytics equivalent] \_trackPageview(opt\_pageURL)
- [Yahoo! Analytics equivalent] submit()

- **trackSiteSearch(keyword, [category], [resultsCount])**

- Log an internal site search for a specific keyword, in an optional category, specifying the optional count of search results in the page.

#### *Configuration of the Tracker Object*

- **setDocumentTitle( string )**

- Override document.title

- [Yahoo! Analytics equivalent] YWATracker.setDocumentName("xxx")

- **setDomains( array )**

- Set array of hostnames or domains to be treated as local. For wildcard subdomains, you can use:

```
setDomains('.example.com');
```

or

```
setDomains('*.*example.com');
```

- [Google Analytics equivalent] `_setDomainName(".example.com")`
- [Yahoo! Analytics equivalent] `setDomains("*.abc.net")`
- **`setCustomUrl( string )`**

- Override the page's reported URL

- **`setReferrerUrl( string )`**

- Override the detected Http-Referer

- **`setSiteId( integer )`**

- Specify the website ID. Redundant: can be specified in

**`getTracker()`**

constructor.

- **`setTrackerUrl( string )`**

- Specify the Piwik server URL. Redundant: can be specified in

**`getTracker()`**

constructor.

- **`setDownloadClasses( string | array )`**

- Set classes to be treated as downloads (in addition to piwik\_download)

- **`setDownloadExtensions( string )`**

- Set list of file extensions to be recognized as downloads. Example: 'doc|pdf|txt'

- **`addDownloadExtensions( string )`**

- Specify additional file extensions to be recognized as downloads. Example: 'doc|pdf|txt'

- **`setIgnoreClasses( string | array )`**

- Set classes to be ignored if present in link (in addition to piwik\_ignore)

- **`setLinkClasses( string | array )`**

- Set classes to be treated as outlinks (in addition to piwik\_link)

- **setLinkTrackingTimer( integer )**

- Set delay for link tracking in milliseconds.

- **discardHashTag( bool )**

- Set to true to not record the hash tag (anchor) portion of URLs

- **setGenerationTimeMs(generationTime)**

- By default Piwik uses the browser DOM Timing API to accurately determine the time it takes to generate and download the page. You may overwrite the value by specifying a milliseconds value here.

- **appendToTrackingUrl(appendToUrl)**

- Appends a custom string to the end of the HTTP request to piwik.php?

- **setDoNotTrack( bool )**

- Set to true to not track users who opt out of tracking using Mozilla's (proposed) Do Not Track setting.

- **disableCookies()**

- Disables all first party cookies. Existing Piwik cookies for this websites will be deleted on the next page view.

- **deleteCookies()**

- Deletes the tracking cookies currently currently set (this is useful when[creating new visits](#))

- **killFrame()**

- Enables a frame-buster to prevent the tracked web page from being framed/iframed.

- **redirectFile( url )**

- Forces the browser load the live URL if the tracked web page is loaded from a local file (e.g., saved to someone's desktop).

- **setHeartBeatTimer( minimumVisitLength, heartBeatDelay )**

- records how long the page has been viewed if the minimumVisitLength (in

seconds) is attained; the heartBeatDelay determines how frequently to update the server

- **getVisitorId()**

- returns the 16 characters ID for the visitor

- **getVisitorInfo()**

- returns the visitor cookie contents in an array

- **getAttributionInfo()**

- returns the visitor attribution array (Referer information and / or Campaign name & keyword).

Attribution information is by Piwik to credit the correct referrer ([first or last referrer](#)) to any goal conversion.

You can also use any of the following functions to get specific attributes of data:

- `piwikTracker.getAttributionCampaignName()`
- `piwikTracker.getAttributionCampaignKeyword()`
- `piwikTracker.getAttributionReferrerTimestamp()`
- `piwikTracker.getAttributionReferrerUrl()`

- **setCustomVariable (index, name, value, scope)**

- Set a custom variable.

- **deleteCustomVariable (index, scope )**

- Delete a custom variable.

- **getCustomVariable (index, scope )**

- Retrieve a custom variable.

- **setCampaignNameKey(name)**

- Set campaign name parameter(s). (Help: [Customize Campaign name parameter names](#))

- **setCampaignKeywordKey(keyword)**

- Set campaign keyword parameter(s). (Help: [Customize Campaign keyword parameter names](#))

- `setConversionAttributionFirstReferrer( bool )`

- Set to true to attribute a conversion to the first referrer. By default, conversion is attributed to the most recent referrer.

#### *Configuration of Tracking Cookies*

Starting with Piwik 1.2, first party cookies are used. Consideration must be given to retention times and avoiding conflicts with other cookies, trackers, and apps.

- `setCookieNamePrefix( prefix )`

- the default prefix is '`pk`'.

- `setCookieDomain( domain )`

- the default is the document domain; if your web site can be visited at both `www.example.com` and `example.com`, you would use:

```
tracker.setCookieDomain('.example.com');
```

or

```
tracker.setCookieDomain('*.*example.com');
```

- `setCookiePath( path )`

- the default is '/'.

- `setVisitorCookieTimeout( seconds )`

- the default is 2 years

- `setSessionCookieTimeout( seconds )`

- the default is 30 minutes

- `setReferralCookieTimeout( seconds )`

- the default is 6 months

## Unit Tests Covering piwik.js

The Piwik JavaScript Tracking API is covered by an extensive JavaScript unit test suite to ensure that the code quality is as high as possible, and that we never break this

functionality. Tests are written using QUnit. To run the tests, simply checkout the [Piwik Git repository](#) and go to

```
/path/to/piwik/tests/javascript/
```

. Tests are run inside your browser.

The Piwik JavaScript API has been tested with numerous web browsers. To maximize coverage, we use services like [crossbrowsertesting.com](#) and [browsershots.org](#).

## Minify piwik.js

The piwik.js is minified to reduce the size that your website visitors will have to download. The YUI Compressor is used to minify the JavaScript ([more information](#)). You can find the original non minified version in [/js/piwik.js](#).

## Frequently Asked Questions

If you have any question about JavaScript Tracking in Piwik, [please search the website](#), or [ask in the forums](#). Enjoy!

- [How do enable tracking for users without Javascript?](#)
- [How does Piwik track downloads?](#)
- [How to track error pages and get the list of 404 and referrers urls.](#)
- [How can I set custom groups of pages \(structure\) so that page view are aggregated by categories?](#)
- [How do I setup Piwik to track multiple websites without revealing the Piwik server URL footprint in JS?](#)
- [How do I disable all tracking cookies used by Piwik in the javascript code?](#)

## The Tracking HTTP API

To track page views, events, visits, you have to send a HTTP request to your Tracking HTTP API endpoint, for example, **http://your-piwik-domain.tld/piwik.php** with the correct query parameters set.

## Supported Query Parameters

This section lists the various query parameters that are supported by the Tracking API. The data for some of these fields will not be available in your app / software which is expected, but you should provide as much information as you can.

*Note: all parameters values that are strings (such as 'url', 'action\_name', etc.) must be URL encoded.*

- Required parameters

- **idsite**

**(required)** — The ID of the website we're tracking a visit/action for.

- **rec**

**(required)** — Required for tracking, must be set to one, eg,

- &rec=1**

- **url**

**(required)** — The full URL for the current action.

- Recommended parameters

- **action\_name**

**(recommended)** — The title of the action being tracked. It is possible to [use slashes / to set one or several categories for this action](#). For example, **Help / Feedback** will create the Action **Feedback** in the category **Help**.

- **\_id**

**(recommended)** — The unique visitor ID, must be a 16 characters hexadecimal string. Every unique visitor must be assigned a different ID and this ID must not change after it is assigned. If this value is not set Piwik will still track visits, but the unique visitors metric might be less accurate.

- **rand**

**(recommended)** — Meant to hold a random value that is generated before each request. Using it helps avoid the tracking request being cached by the browser or a proxy.

- **apiv**

**(recommended)** — The parameter &apiv=1 defines the api version to use (currently always set to 1)

- Optional visitor info (*We recommend that these parameters be used if the information is available and relevant to your use case.*)

- urlref

— The full HTTP Referrer URL. This value is used to determine how someone got to your website (ie, through a website, search engine or campaign).

- \_cvar

— Visit scope [custom variables](#). This is a JSON encoded string of the custom variable array (see below for an example value).

- \_idvc

— The current count of visits for this visitor. To set this value correctly, it would be required to store the value for each visitor in your application (using sessions or persisting in a database). Then you would manually increment the counts by one on each new visit or "session", depending on how you choose to define a visit. This value is used to populate the report *Visitors > Engagement > Visits by visit number*.

- \_viewts

— The UNIX timestamp of this visitor's previous visit. This parameter is used to populate the report *Visitors > Engagement > Visits by days since last visit*.

- \_idts

— The UNIX timestamp of this visitor's first visit. This could be set to the date where the user first started using your software/app, or when he/she created an account. This parameter is used to populate the *Goals > Days to Conversion* report.

- \_rcn

— The Campaign name (see [Tracking Campaigns](#)). Used to populate the *Referrers > Campaigns* report. Note: *this parameter will only be used for the first pageview of a visit*.

- \_rck**

— The Campaign Keyword (see [Tracking Campaigns](#)). Used to populate the *Referrers > Campaigns* report (clicking on a campaign loads all keywords for this campaign). *Note: this parameter will only be used for the first pageview of a visit.*

- res**

— The resolution of the device the visitor is using, eg **1280x1024**.

- h**

— The current hour (local time).

- m**

— The current minute (local time).

- s**

— The current second (local time).

- ua**

— An override value for the **User-Agent** HTTP header field. The user agent is used to detect the operating system and browser used.

- lang**

— An override value for the **Accept-Language** HTTP header field. This value is used to detect the visitor's country if [GeolP](#) is not enabled.

- Optional action/event info

- cvar**

— Page scope [custom variables](#). This is a JSON encoded string of the custom variable array (see below for an example value).

- link**

— An external URL the user has opened. Used for tracking outlink clicks. We recommend to also set the **url** parameter to this same value.

- download**

— URL of a file the user has downloaded. Used for tracking downloads. We recommend to also set the **url** parameter to this same value.

- **search**

— The Site Search keyword. When specified, the request will not be tracked as a normal pageview but will instead be tracked as a [Site Search](#) request.

- **search\_cat**

— when **search** is specified, you can optionally specify a search category with this parameter.

- **search\_count**

— when **search** is specified, we also recommend to set the **search\_count** to the number of search results displayed on the results page. When keywords are tracked with &search\_count=0 they will appear in the "No Result Search Keyword" report.

- **idgoal**

— If specified, the tracking request will trigger a conversion for the goal of the website being tracked with this ID.

- **revenue**

— A monetary value that was generated as revenue by this goal conversion. Only used if **idgoal** is specified in the request.

- **gt\_ms**

— The amount of time it took the server to generate this action, in milliseconds. This value is used to process the **Avg. generation time** column in the Page URL and Page Title reports, as well as a site wide running average of the speed of your server.*Note: when using the Javascript tracker this value is set to the time for server to generate response + the time for client to download response.*

- Special parameters

The following parameters require that you set

- &**token\_auth**=

to the token\_auth value of the Super User or a user with admin access to the website visits are being tracked for.

- **token\_auth**

— 32 character authorization key used to authenticate the API request.

- **cip**

— Override value for the visitor IP (both IPv4 and IPv6 notations supported).

- **cdt**

— Override for the datetime of the request (normally the current time is used). This can be used to record visits and page views in the past. The expected format is:

**2011-04-05 00:11:42**

(remember to URL encode the value!). *Note: if you record data in the past, you will need to [force Piwik to re-process reports for the past dates](#).*

- **cid**

— defines the visitor ID for this request. You must set this value to exactly a 16 character hexadecimal string (containing only characters 01234567890abcdefABCDEF). When specified, the Visitor ID will be "enforced". This means that if there is no recent visit with this visitor ID, a new one will be created. If a visit is found in the last 30 minutes with your specified Visitor Id, then the new action will be recorded to this existing visit.

- **new\_visit**

— If set to 1, will force a new visit to be created for this action. This feature is also [available in Javascript](#).

- **country**

— An override value for the country. Should be set to the two letter country code of the visitor (lowercase), eg **fr**, **de**, **us**.

- **region**

— An override value for the region. Should be set to the two letter region code as defined by [MaxMind's](#) GeoIP databases. See [here](#) for a list of them for every country (the region codes are located in the second column, to the left of the region name and to the right of the country code).

- **city**

— An override value for the city. The name of the city the visitor is located in, eg, **Tokyo**.

- **lat**

— An override value for the visitor's latitude, eg 22.456.

- **long**

— An override value for the visitor's longitude, eg 22.456.

## Tracking Bots

By default Piwik does not track bots. If you use the Tracking HTTP API directly, you may be interested in tracking bot requests. To enable Bot Tracking in Piwik, set the parameter **&bots=1** in your requests to piwik.php.

## Example Tracking Request

Here is an example of a real tracking request used by the [Piwik Mobile app](#) when anonymously tracking Mobile App usage:

```
http://piwik-server/piwik.php?_cvar={"1":["OS","iphone 5.0"],"2":["Piwik Mobile Version","1.6.2"],"3":["Locale","en:en"],"4":["Num Accounts","2"]}&action_name=View settings&url=http://mobileapp.piwik.org/window/settings &idsite=8876&rand=351459&h=18&m=13&s=3 &rec=1&apiv=1&cookie= &urlref=http://iphone.mobileapp.piwik.org&_id=af344a398df83874 &_idvc=19&res=320 脳 480&
```

*Note: for clarity, parameter values are not URL encoded in this example.*

**Explanation:** this URL has custom variables for the OS, Piwik version, number of accounts created. It tracks an event named **View settings** with a fake URL, records the screen resolution and also includes a custom unique ID generated to ensure all requests for the same Mobile App user will be recorded for the same visit in Piwik.

## Bulk Tracking

Some applications such as the [Piwik log importer](#), have to track many visits, sometimes tens, hundreds, thousands or even more all at once. Tracking these requests with one HTTP request per visit or action can result in *enormous* delays due to the amount of time it takes to send an HTTP request. Using the bulk tracking feature, however, these requests can be sent all at once making the application far more efficient.

To send a bulk tracking request, an HTTP POST must be made with a JSON object to the Piwik tracking endpoint. The object must contain the following properties:

- **token\_auth**

— token\_auth can be found in the API page. Provide at least Admin or Super User permission to use

- **requests**

— an array of individual tracking requests. Each tracking request should be the query string you'd send if you were going to track that action individually.

## Example Requests

This is an example of the payload of a bulk tracking request:

```
{  
    "requests": [  
        "?idsite=1&url=http://example.org&action_name=Test bulk log Pageview&rec=1",  
        "?idsite=1&url=http://example.net/test.htm&action_name=Another bulk page view&rec=1"  
    ],  
    "token_auth": "33dc3f2536d3025974cccb4b4d2d98f4"  
}
```

It can be sent to Piwik using curl with the following command:

```
curl -i -X POST -d '{"requests": ["?idsite=1&url=http://example.org&action_name=Test bulk log Pageview&rec=1", "?idsite=1&url=http://example.net/test.htm&action_name=Another bulk page view&rec=1"], "token_auth": "33dc3f2536d3025974cccb4b4d2d98f4"}' http://piwik.example.com/piwik.php
```

This will track **two** actions using only **one** HTTP request to Piwik.

# Debugging the Tracker

To verify that your data is being tracked properly, you can enable debug logging in the Piwik tracking file, **piwik.php**.

**Tracking requests will then output the tracking log messages rather than displaying a 1\*1 transparent GIF beacon.**

Follow these steps to enable debug logging for the tracker:

1. In the file

```
path/to/piwik/piwik.php
```

, you can set

```
$GLOBALS['PIWIK_TRACKER_DEBUG'] = true;
```

2. Look at the HTTP requests that are sent to Piwik.

- If the requests take place in a browser, you can use a tool like the [Firebug](#) to see all requests to **piwik.php**.
- If the requests are triggered from your app or software directly, you can output or log the output of tracking requests and to view the debug messages.

## Learn more

- For a **list of tracking clients** see this [page](#).
- To learn more **about geolocation** read the [GeoIP user docs](#).
- To learn **about Piwik's JavaScript tracker** read our [documentation for the tracker](#).

# Reporting API Reference

This is the Piwik API Reference. It lists all functions that can be called, documents the parameters, and links to examples for every call in the various formats.

# API Request

## Standard API parameters

- **idSite**
  - the integer id of your website, eg. `idSite=1`
  - you can also specify a list of idSites comma separated, eg. `idSite=1,4,5,6`
  - if you want to get data for all websites, set `idSite=all`
- **period** — the period you request the statistics for. Can be any of: `day`, `week`, `month`, `year` or `range`. All reports are returned for the dates based on the website's time zone.
  - **day** returns data for a given day.
  - **week** returns data for the week that contains the specified 'date'
  - **month** returns data for the month that contains the specified 'date'
  - **year** returns data for the year that contains the specified 'date'
  - **range** returns data for the specified 'date' range.

For example to request a report for the range Jan 1st to Feb 15th you would write `&period=range&date=2011-01-01,2011-02-15`

- **date**
  - standard format = `YYYY-MM-DD`
  - magic keywords = `today` or `yesterday`. These are relative to the website timezone. For example, for a website with UTC+12 timezone, "date=today" for an API request at 5PM UTC on 2010-01-01 will return the reports for 2010-01-02.
  - range of dates
    - `/lastX` for the last X periods including today (eg `&date=last10&period=day` would return an entry for each of the last 10 days including today). This is relative to the website timezone.
    - `previousX` returns the last X periods before today (eg. `&date=last52&period=week` will return an entry for each of the 52 weeks before this week). This is relative to the website timezone.
    - `YYYY-MM-DD,YYYY-MM-DD` for every period (day, week, month or year) in the date range
    - Note: if you set '`period=range`' to request data for a custom date range, the API will return the sum of data for the specified date

range. When 'period=range', the following keywords are supported for the parameter 'date':

- *lastX*
- *previousX*
- *YYYY-MM-DD, YYYY-MM-DD,*  
or *YYYY-MM-DD,today* or *YYYY-MM-DD,yesterday*
- **segment** — defines the Custom Segment you wish to filter your reports to.
  - for example, 'referrerName==twitter.com' will return the requested API report, processed for the subset of users coming from twitter.com
  - segments can be combined in AND and OR operations. For example, to filter for "Visits where (Referrer name is Google OR Referrer name is Bing) AND Country is India", you would write: `referrerName==Google,referrerName==Bing;country==IN`
    - see [segmentation documentation](#) for the list of available dimensions & metrics, example values for each, and more information about the custom segment parameter
- **format**; defines the format of the output
  - xml
  - json (if you want to do [cross domain request in ajax](#) and get json data, you can wrap the json data around a function call by using the **jsoncallback** parameter)
  - csv (comma-separated values)
  - tsv (tab-separated values, similar to CSV but loads properly in Excel)
  - html
  - php; when you export in PHP format it is serialized by default (set `serialize=0` to get the raw php data structure). You can have a visual output of the data by setting `prettyDisplay=1`
  - rss (when **date** is a range for example `date=last10` or `date=previous15`)
  - original; to fetch the original PHP data structure. This is useful when you call the Piwik API [internally using the PHP code](#)

## Optional API parameters

Each API call can contain parameters that do not appear in the list of parameters, but act as "filters". Filters can be presentation filters (eg. specify the language for internationalization), or act as data helpers (sort results, search for a dataset subset, fetch children of a given entity).

Here is an overview of the parameters you can add to any API request, where applicable:

- **language**; if specified, returns data strings that can be internationalized and will be translated. For example, dates and times returned by the Live API can be translated into the specified language. Expected value is the 2 language letters code, eg. en, fr, de, es, etc. You can get the available list of language by calling the LanguagesManager API.
- **idSubtable** is used to request a subtable of a given row. In Piwik, some rows are linked to a sub-table. For example, each row in the Referers.getSearchEngines response have an "idsubdatatable" field. This integer idsubdatatable is the idSubtable of the table that contains all keywords for this search engine. You can then request the keywords for this search engine by calling Referers.getKeywordsFromSearchEngineld with the parameter idSubtable=X (replace X with the idsubdatatable value found in the Referers.getSearchEngines response, for the search engine you are interested in).
- **expanded**; some API functions have a parameter 'expanded'. If 'expanded' is set to 1, the returned data will contain the first level results, as well as all sub-tables. See, for example, the[returned dataset for the Actions.getDownloads API function](#).
- **flat**; some API functions have a parameter 'expanded', which means that the data is hierarchical. For such API function, if 'flat' is set to 1, the returned data will contain the flattened view of the table data set. The children of all first level rows will be aggregated under one row. This is useful for example to see all Custom Variables names and values at once, for example, [Piwik forum user status](#), or to see the full URLs not broken down by directory or structure.
- **label**; this parameter can be used to search only for the row matching a given label. When specified, the report data will be filtered and return only the rows where the row label matches the specified parameter. For example you can set &label=Nice%20Keyword to keep only the row with a label "Nice Keyword". There are also generic filters you can choose to apply on all APIs that return web analytics reports. For example, there is a filter for sorting by column, define start and number of rows to return, a filter to only return rows matching a given string,
- **filter\_offset**; defines the offset of the starting row being returned
- **filter\_limit**; defines the number of rows to be returned. Set to -1 to return all rows. By default, only the top 100 rows are returned.
- **filter\_truncate**; if set, will truncate the table after \$filter\_truncate rows. The last row will be named 'Others' (localized in the requested language) and the columns will be an aggregate of statistics of all truncated rows.
- **filter\_pattern**; defines the text you want to search for in the **filter\_column**. Only the row with the given column matching the pattern will be returned.
- **filter\_column** ; defines the column that we want to search for a text (see **filter\_pattern**). If not specified, defaults to 'label' (from Piwik 1.7)
- **filter\_sort\_order**; defines the order of the results, asc or desc

- **filter\_sort\_column**; defines the column to be sorted by
- **filter\_excludelowpop**; defines the column to use for the threshold of value**filter\_excludelowpop\_value**; only the columns with a value greater than**filter\_excludelowpop\_value**; will be returned
- **filter\_excludelowpop\_value**; defines the minimum value for the **filter\_excludelowpopcolumn**
- **hideColumns**; a comma separated list of columns. If set, removes those columns from the result. This can be used to reduce the amount of data transferred.
- **showColumns**; a comma separated list of columns. If set, removes all columns in the result that are not found in this list. This can be used to reduce the amount of data transferred.
- **filter\_column\_recursive**; defines the column to be searched for when recursively searching for a pattern *filter\_pattern\_recursive*
- **filter\_pattern\_recursive**; defines the text you are searching for. Only the matching rows are returned. This filter is applied to recursive tables (Actions/Downloads/Outlinks tables)
- **disable\_generic\_filters**; if set to 1, all the generic filters above will not be applied. This can be useful to disable the filters above which are otherwise applied with default values.
- **disable\_queued\_filters**; if set to 1, all the filters that are mostly presentation filters (replace a column name, apply callbacks on the column to add new information such as the browser icon URL, etc.) will not be applied.

## Passing an array of data as a parameter

Some parameters can optionally accept arrays. For example, the urls parameter of `SitesManager.addSite`, `SitesManager.addSiteAliasUrls`, and `SitesManager.updateSite` allows for an array of urls to be passed. To pass an array add the bracket operators and an index to the parameter name in the get request. So, to call `SitesManager.addSite` with two urls you would use the following array:

[http://demo.piwik.org/?module=API&method=SitesManager.addSite&siteName=new%20example%20website&urls\[0\]=http://example.org&urls\[1\]=http://example-alias.org](http://demo.piwik.org/?module=API&method=SitesManager.addSite&siteName=new%20example%20website&urls[0]=http://example.org&urls[1]=http://example-alias.org)

## Advanced Users: Send multiple API Requests at once

Sometimes it is necessary to call the Piwik API a few times to get the data needed for a report or custom application. When you need to call many API functions simultaneously or

if you just don't want to issue a lot of HTTP requests, you may want to consider using a **Bulk API Request**. This feature allows you to call several API methods with one HTTP request (either a GET or POST).

To issue a bulk request, call the API.getBulkRequest method and pass the API methods & parameters (each request must be [URL Encoded](#)) you wish to call in the 'urls' query parameter. For example, to call VisitsSummary.get & VisitorInterest.getNumberOfVisitsPerVisitDuration at the same time, you can use:

```
http://demo.piwik.org/?module=API&method=API.getBulkRequest&format=json&urls[0]=method%3dVisitsSummary.get%26idSite%3d1%26date%3d2012-03-06%26period%3dday&urls[1]=method%3dVisitorInterest.getNumberOfVisitsPerVisitDuration%26idSite%3d1%26date%3d2012-03-06%26period%3dday
```

Notice that urls[0] is the url-encoded call to VisitsSummary.get by itself and that urls[1] is what you would use to call VisitorInterest.getNumberOfVisitsPerVisitDuration by itself. The &format is specified only once (format=xml and format=json are supported for bulk requests).

The API Response will be an array containing the formatted result of each individual API method, in this case VisitsSummary.get and VisitorInterest.getNumberOfVisitsPerVisitDuration.

## Authenticate to the API via token\_auth parameter

In the example above, the request works because the statistics are public (the *anonymous* user has a *view* access to the website). By default in Piwik your statistics are private. In the case that you cannot have your statistics to be public:

- when you access your Piwik installation you are requested to log in
- when you call the API over http you need to authenticate yourself This is done by adding a secret parameter in the URL. This parameter is as secret as your login and password!

You can get this token in the *Manage Users* admin area.

Then you simply have to add the parameter **&token\_auth=YOUR\_TOKEN** at the end of your API call URL.

## API Response: Metric Definitions

Here is a list of metrics returned by the API and their definition.

## General Metrics

- nb\_uniq\_visitors - Number of unique visitors
- nb\_visits - Number of Visits (30 min of inactivity considered a new visit)
- nb\_actions - Number of actions (page views, outlinks and downloads)
- sum\_visit\_length - Total time spent, in seconds
- bounce\_count - Number of visits that bounced (viewed only one page)
- max\_actions - Maximum number of actions in a visit
- nb\_visits\_converted - Number of visits that converted a goal
- nb\_conversions - Number of goal conversions
- revenue - Total revenue of goal conversions

## Metrics Specific to Page URLs and Page Titles Reports

- nb\_hits - Number of views on this page
- entry\_nb\_visits - Number of visits that started on this page
- entry\_nb\_uniq\_visitors - Number of unique visitors that started their visit on this page
- entry\_nb\_actions - Number of page views for visits that started on this page
- entry\_sum\_visit\_length - Time spent, in seconds, by visits that started on this page
- entry\_bounce\_count - Number of visits that started on this page, and bounced (viewed only one page)
- exit\_nb\_visits - Number of visits that finished on this page
- exit\_nb\_uniq\_visitors - Number of unique visitors that ended their visit on this page
- sum\_time\_spent - Total time spent on this page, in seconds
- sum\_daily\_nb\_uniq\_visitors - Sum of daily unique visitors over days in the period. Piwik doesn't process unique visitors across the full period.
- sum\_daily\_entry\_nb\_uniq\_visitors - Sum of daily unique visitors that started their visit on this page
- sum\_daily\_exit\_nb\_uniq\_visitors - (deprecated) Same as sum\_daily\_entry\_nb\_uniq\_visitors
- exit\_bounce\_count - (deprecated) Same as entry\_bounce\_count

## Processed metrics, appearing in the Metadata API response

- avg\_time\_on\_page - Average time spent, in seconds, on this page
- bounce\_rate - Ratio of visitors leaving the website after landing on this page
- exit\_rate - Ratio of visitors that do not view any other page after this page

Ecommerce metrics, appearing in the Ecommerce Goals.getItems\*  
API calls only:

- revenue - The total revenue generated by Product sales. Excludes tax, shipping and discount.
- quantity - Quantity is the total number of products sold for each Product SKU/Name/Category.
- orders - It is the total number of Ecommerce orders which contained this Product SKU/Name/Category at least once.
- abandoned\_carts - This value is only set if the request contains '&abandonedCarts=1'. In this case, "orders" metrics will not be returned. It is the total number of abandoned carts which contained this Product SKU/Name/Category at least once.
- avg\_price - The average revenue for this Product/Category.
- avg\_quantity - The average quantity for this Product/Category.
- nb\_visits - This value appears only if you have set up '[Ecommerce Product/Category page tracking](#)'. The number of visits on the Product/Category page.
- conversion\_rate - This value appears only if you have set up '[Ecommerce Product/Category page tracking](#)'. The conversion rate is the number of orders (or abandoned\_carts if the request contains '&abandonedCarts=1') containing this product/category divided by number of visits on the product/category page.

## API Method List

### Quick access to APIs

[API](#)

[Actions](#)

[Annotations](#)

[CustomVariables](#)

[Dashboard](#)

[DevicesDetection](#)

[ExampleAPI](#)

[ExamplePlugin](#)

[Goals](#)

[ImageGraph](#)

[LanguagesManager](#)

[Live](#)

[MobileMessaging](#)

[MultiSites](#)

[Overlay](#)

[Provider](#)  
[Referrers](#)  
[SEO](#)  
[ScheduledReports](#)  
[SegmentEditor](#)  
[SitesManager](#)  
[Transitions](#)  
[UserCountry](#)  
[UserSettings](#)  
[UsersManager](#)  
[VisitFrequency](#)  
[VisitTime](#)  
[VisitorInterest](#)  
[VisitsSummary](#)

## Module API

This API is the [Metadata API](#): it gives information about all other available APIs methods, as well as providing human readable and more complete outputs than normal API methods. Some of the information that is returned by the Metadata API:

- the dynamically generated list of all API methods via "getReportMetadata"
- the list of metrics that will be returned by each method, along with their human readable name, via "getDefaultMetrics" and "getDefaultProcessedMetrics"
- the list of segments metadata supported by all functions that have a 'segment' parameter
- the (truly magic) method "getProcessedReport" will return a human readable version of any other report, and include the processed metrics such as conversion rate, time on site, etc. which are not directly available in other methods.
- the method "getSuggestedValuesForSegment" returns top suggested values for a particular segment. It uses the Live.getLastVisitsDetails API to fetch the most recently used values, and will return the most often used values first.

The Metadata API is for example used by the Piwik Mobile App to automatically display all Piwik reports, with translated report & columns names and nicely formatted values. More information on the [Metadata API documentation page](#) @method static \Piwik\Plugins\API\API getInstance()

- **API.getPiwikVersion** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **API.getSettings** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **API.getDefaultMetricTranslations** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **API.getSegmentsMetadata** (idSites = 'Array') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **API.getLogoUrl** (pathOnly = '') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **API.getHeaderLogoUrl** (pathOnly = '') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **API.getMetadata** (idSite, apiModule, apiAction, apiParameters = 'Array', language = '',

period = "", date = "", hideMetricsDoc = "", showSubtableReports = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **API.getReportMetadata** (idSites = "", period = "", date = "", hideMetricsDoc = "", showSubtableReports = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **API.getProcessedReport** (idSite, period, date, apiModule, apiAction, segment = "", apiParameters = "", idGoal = "", language = "", showTimer = '1', hideMetricsDoc = "", idSubtable = "", showRawMetrics = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **API.get** (idSite, period, date, segment = "", columns = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **API.getRowEvolution** (idSite, period, date, apiModule, apiAction, label = "", segment = "", column = "", language = "", idGoal = "", legendAppendMetric = '1', labelUseAbsoluteUrl = '1') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **API.getBulkRequest** (urls) [ No example available ]  
- **API.getSuggestedValuesForSegment** (segmentName, idSite) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module Actions

The Actions API lets you request reports for all your Visitor Actions: Page URLs, Page titles (Piwik Events), File Downloads and Clicks on external websites. For example, "getPageTitles" will return all your page titles along with standard [Actions metrics](#) for each row. It is also possible to request data for a specific Page Title with "getPageTitle" and setting the parameter `pageName` to the page title you wish to request. Similarly, you can request metrics for a given Page URL via "getPageUrl", a Download file via "getDownload" and an outlink via "getOutlink". Note: `pageName`, `pageUrl`, `outlinkUrl`, `downloadUrl` parameters must be URL encoded before you call the API. @method static \Piwik\Plugins\Actions\API getInstance()

- **Actions.get** (idSite, period, date, segment = "", columns = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **Actions.getPageUrls** (idSite, period, date, segment = "", expanded = "", idSubtable = "", depth = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **Actions.getPageUrlsFollowingSiteSearch** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **Actions.getPageTitlesFollowingSiteSearch** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **Actions.getEntryPageUrls** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **Actions.getExitPageUrls** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]  
- **Actions.getPageUrl** (pageUrl, idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **Actions.getPageTitles** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getEntryPageTitles** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getExitPageTitles** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getPageTitle** (pageName, idSite, period, date, segment = "") [ No example available ]
- **Actions.getDownloads** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getDownload** (downloadUrl, idSite, period, date, segment = "") [ No example available ]
- **Actions.getOutlinks** (idSite, period, date, segment = "", expanded = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getOutlink** (outlinkUrl, idSite, period, date, segment = "") [ No example available ]
- **Actions.getSiteSearchKeywords** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getSiteSearchNoResultKeywords** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Actions.getSiteSearchCategories** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module Annotations

API for annotations plugin. Provides methods to create, modify, delete & query annotations. @method static \Piwik\Plugins\Annotations\API getInstance()

- **Annotations.add** (idSite, date, note, starred = '0') [ No example available ]
- **Annotations.save** (idSite, idNote, date = "", note = "", starred = "") [ No example available ]
- **Annotations.delete** (idSite, idNote) [ No example available ]
- **Annotations.get** (idSite, idNote) [ No example available ]
- **Annotations.getAll** (idSite, date = "", period = 'day', lastN = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Annotations.getAnnotationCountForDates** (idSite, date, period, lastN = "", getAnnotationText = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module CustomVariables

The Custom Variables API lets you access reports for your [Custom Variables](#) names and values. @method static \Piwik\Plugins\CustomVariables\API getInstance()

- **CustomVariables.getCustomVariables** (idSite, period, date, segment = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **CustomVariables.getCustomVariablesValuesFromNameId** (idSite, period, date, idSubtable, segment = "") [ No example available ]

[↑ Back to top](#)

## Module Dashboard

This API is the [Dashboard API](#): it gives information about dashboards. @method static \Piwik\Plugins\Dashboard\API getInstance()

- **Dashboard.getDashboards** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module DevicesDetection

The DevicesDetection API lets you access reports on your visitors devices, brands, models, Operating system, Browsers. @method static \Piwik\Plugins\DevicesDetection\API getInstance()

- **DevicesDetection.getType** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **DevicesDetection.getBrand** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **DevicesDetection.getModel** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **DevicesDetection.getOsFamilies** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **DevicesDetection.getOsVersions** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **DevicesDetection.getBrowserFamilies** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **DevicesDetection.getBrowserVersions** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module ExampleAPI

The ExampleAPI is useful to developers building a custom Piwik plugin. Please see the [source code in the file plugins/ExampleAPI/API.php](#) for more documentation.

@method static \Piwik\Plugins\ExampleAPI\API getInstance()

- **ExampleAPI.getPiwikVersion** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getAnswerToLife** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getObject** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getSum** (a = '0', b = '0') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getNull** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getDescriptionArray** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

- **ExampleAPI.getCompetitionDatatable** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getMoreInformationAnswerToLife** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **ExampleAPI.getMultiArray** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module ExamplePlugin

API for plugin ExamplePlugin @method static \Piwik\Plugins\ExamplePlugin\API getInstance()

- **ExamplePlugin.getAnswerToLife** (truth = '1') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module Goals

Goals API lets you Manage existing goals, via "updateGoal" and "deleteGoal", create new Goals via "addGoal", or list existing Goals for one or several websites via "getGoals" If you are [tracking Ecommerce orders and products](#) on your site, the functions "getItemsSku", "getItemsName" and "getItemsCategory" will return the list of products purchased on your site, either grouped by Product SKU, Product Name or Product Category. For each name, SKU or category, the following metrics are returned: Total revenue, Total quantity, average price, average quantity, number of orders (or abandoned carts) containing this product, number of visits on the Product page, Conversion rate. By default, these functions return the 'Products purchased'. These functions also accept an optional parameter &abandonedCarts=1. If the parameter is set, it will instead return the metrics for products that were left in an abandoned cart therefore not purchased. The API also lets you request overall Goal metrics via the method "get": Conversions, Visits with at least one conversion, Conversion rate and Revenue. If you wish to request specific metrics about Ecommerce goals, you can set the parameter &idGoal=ecommerceAbandonedCart to get metrics about abandoned carts (including Lost revenue, and number of items left in the cart) or &idGoal=ecommerceOrder to get metrics about Ecommerce orders (number of orders, visits with an order, subtotal, tax, shipping, discount, revenue, items ordered) See also the documentation about [Tracking Goals](#) in Piwik. @method static \Piwik\Plugins\Goals\API getInstance()

- **Goals.getGoals** (idSite) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **Goals.addGoal** (idSite, name, matchAttribute, pattern, patternType, caseSensitive = "", revenue = "", allowMultipleConversionsPerVisit = "") [ No example available ]
- **Goals.updateGoal** (idSite, idGoal, name, matchAttribute, pattern, patternType, caseSensitive = "", revenue = "", allowMultipleConversionsPerVisit = "") [ No example available ]
- **Goals.deleteGoal** (idSite, idGoal) [ No example available ]
- **Goals.getItemsSku** (idSite, period, date, abandonedCarts = "", segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Goals.getItemsName** (idSite, period, date, abandonedCarts = "", segment = "") [ Example

in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **Goals.getItemsCategory** (idSite, period, date, abandonedCarts = "", segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Goals.get** (idSite, period, date, segment = "", idGoal = "", columns = 'Array') [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Goals.getDaysToConversion** (idSite, period, date, segment = "", idGoal = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Goals.getVisitsUntilConversion** (idSite, period, date, segment = "", idGoal = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module ImageGraph

The ImageGraph.get API call lets you generate beautiful static PNG Graphs for any existing Piwik report. Supported graph types are: line plot, 2D/3D pie chart and vertical bar chart. A few notes about some of the parameters available:

- \$graphType defines the type of graph plotted, accepted values are: 'evolution', 'verticalBar', 'pie' and '3dPie'
  - \$colors accepts a comma delimited list of colors that will overwrite the default Piwik colors
  - you can also customize the width, height, font size, metric being plotted (in case the data contains multiple columns/metrics). See also [How to embed static Image Graphs?](#) for more information.
- @method static \Piwik\Plugins\ImageGraph\API getInstance()
- **ImageGraph.get** (idSite, period, date, apiModule, apiAction, graphType = "", outputType = '0', columns = "", labels = "", showLegend = '1', width = "", height = "", fontSize = '9', legendFontSize = "", aliasedGraph = '1', idGoal = "", colors = "", textColor = '222222', backgroundColor = 'FFFFFF', gridColor = 'CCCCCC', idSubtable = "", legendAppendMetric = '1', segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module LanguagesManager

The LanguagesManager API lets you access existing Piwik translations, and change Users languages preferences. "getTranslationsForLanguage" will return all translation strings for a given language, so you can leverage Piwik translations in your application (and automatically benefit from the [40+ translations!](#)). This is mostly useful to developers who integrate Piwik API results in their own application. You can also request the default language to load for a user via "getLanguageForUser", or update it via "setLanguageForUser". @method static \Piwik\Plugins\LanguagesManager\API getInstance()

- **LanguagesManager.isLanguageAvailable** (languageCode) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **LanguagesManager.getAvailableLanguages** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

- **LanguagesManager.getAvailableLanguagesInfo ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **LanguagesManager.getAvailableLanguageNames ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **LanguagesManager.getTranslationsForLanguage (languageCode)** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **LanguagesManager.getLanguageForUser (login)** [ No example available ]
- **LanguagesManager.setLanguageForUser (login, languageCode)** [ No example available ]

[↑ Back to top](#)

## Module Live

The Live! API lets you access complete visit level information about your visitors. Combined with the power of [Segmentation](#), you will be able to request visits filtered by any criteria. The method "getLastVisitsDetails" will return extensive data for each visit, which includes: server time, visitId, visitorId, visitorType (new or returning), number of pages, list of all pages (and events, file downloaded and outlinks clicked), custom variables names and values set to this visit, number of goal conversions (and list of all Goal conversions for this visit, with time of conversion, revenue, URL, etc.), but also other attributes such as: days since last visit, days since first visit, country, continent, visitor IP, provider, referrer used (referrer name, keyword if it was a search engine, full URL), campaign name and keyword, operating system, browser, type of screen, resolution, supported browser plugins (flash, java, silverlight, pdf, etc.), various dates & times format to make it easier for API users... and more! With the parameter '[&segment=](#)' you can filter the returned visits by any criteria (visitor IP, visitor ID, country, keyword used, time of day, etc.). The method "getCounters" is used to return a simple counter: visits, number of actions, number of converted visits, in the last N minutes. See also the documentation about [Real time widget and visitor level reports](#) in Piwik. @method static \Piwik\Plugins\Live\API getInstance()

- **Live.getCounters (idSite, lastMinutes, segment = "")** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **Live.getLastVisitsDetails (idSite, period = "", date = "", segment = "", numLastVisitorsToFetch = "", minTimestamp = "", flat = "", doNotFetchActions = "")** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Live.getVisitorProfile (idSite, visitorId = "", segment = "", checkForLatLong = "")** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **Live.getMostRecentVisitorId (idSite, segment = "")** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module MobileMessaging

The MobileMessaging API lets you manage and access all the MobileMessaging plugin features including : - manage SMS API credential - activate phone numbers - check remaining credits - send SMS @method static \Piwik\Plugins\MobileMessaging\API getInstance()

- **MobileMessaging.areSMSAPICredentialProvided** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **MobileMessaging.getSMSProvider** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **MobileMessaging.setSMSAPICredential** (provider, apiKey) [ No example available ]
- **MobileMessaging.addPhoneNumber** (phoneNumber) [ No example available ]
- **MobileMessaging.getCreditLeft** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **MobileMessaging.removePhoneNumber** (phoneNumber) [ No example available ]
- **MobileMessaging.validatePhoneNumber** (phoneNumber, verificationCode) [ No example available ]
- **MobileMessaging.deleteSMSAPICredential** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **MobileMessaging.setDelegatedManagement** (delegatedManagement) [ No example available ]
- **MobileMessaging.getDelegatedManagement** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module MultiSites

The MultiSites API lets you request the key metrics (visits, page views, revenue) for all Websites in Piwik. @method static \Piwik\Plugins\MultiSites\API getInstance()

- **MultiSites.getAll** (period, date, segment = "", enhanced = "", pattern = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **MultiSites.getOne** (idSite, period, date, segment = "", enhanced = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module Overlay

Class API @method static \Piwik\Plugins\Overlay\API getInstance()

- **Overlay.getTranslations** (idSite) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **Overlay.getExcludedQueryParameters** (idSite) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **Overlay.getFollowingPages** (url, idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module Provider

The Provider API lets you access reports for your visitors Internet Providers. @method static \Piwik\Plugins\Provider\API getInstance()

- **Provider.getProvider** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

# Module Referrers

The Referrers API lets you access reports about Websites, Search engines, Keywords, Campaigns used to access your website. For example, "getKeywords" returns all search engine keywords (with [general analytics metrics](#) for each keyword), "getWebsites" returns referrer websites (along with the full Referrer URL if the parameter &expanded=1 is set). "getReferrerType" returns the Referrer overview report. "getCampaigns" returns the list of all campaigns (and all campaign keywords if the parameter &expanded=1 is set). The methods "getKeywordsForPageUrl" and "getKeywordsForPageTitle" are used to output the top keywords used to find a page. Check out the widget "[Top keywords used to find this page](#)" that you can easily re-use on your website.

@method static \Piwik\Plugins\Referrers\API getInstance()

- **Referrers.getReferrerType** (idSite, period, date, segment = "", typeReferrer = "", idSubtable = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getAll** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getKeywords** (idSite, period, date, segment = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getKeywordsForPageUrl** (idSite, period, date, url) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getKeywordsForPageTitle** (idSite, period, date, title) [ No example available ]
- **Referrers.getSearchEnginesFromKeywordId** (idSite, period, date, idSubtable, segment = "") [ No example available ]
- **Referrers.getSearchEngines** (idSite, period, date, segment = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getKeywordsFromSearchEngineId** (idSite, period, date, idSubtable, segment = "") [ No example available ]
- **Referrers.getCampaigns** (idSite, period, date, segment = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getKeywordsFromCampaignId** (idSite, period, date, idSubtable, segment = "") [ No example available ]
- **Referrers.getWebsites** (idSite, period, date, segment = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getUrlsFromWebsiteId** (idSite, period, date, idSubtable, segment = "") [ No example available ]
- **Referrers.getSocials** (idSite, period, date, segment = "", expanded = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getUrlsForSocial** (idSite, period, date, segment = "", idSubtable = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getNumberOfDistinctSearchEngines** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **Referrers.getNumberOfDistinctKeywords** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **Referrers.getNumberOfDistinctCampaigns** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **Referrers.getNumberOfDistinctWebsites** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **Referrers.getNumberOfDistinctWebsitesUrls** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module SEO

The SEO API lets you access a list of SEO metrics for the specified URL: Google Pagerank, Goolge/Bing indexed pages Alexa Rank, age of the Domain name and count of DMOZ entries. @method static \Piwik\Plugins\SEO\API getInstance()

- **SEO.getRank** (url) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module ScheduledReports

The ScheduledReports API lets you manage Scheduled Email reports, as well as generate, download or email any existing report. "generateReport" will generate the requested report (for a specific date range, website and in the requested language). "sendEmailReport" will send the report by email to the recipients specified for this report. You can also get the list of all existing reports via "getReports", create new reports via "addReport", or manage existing reports with "updateReport" and "deleteReport". See also the documentation about [Scheduled Email reports](#) in Piwik. @method static \Piwik\Plugins\ScheduledReports\API getInstance()

- **ScheduledReports.addReport** (idSite, description, period, hour, reportType, reportFormat, reports, parameters, idSegment = "") [ No example available ]

- **ScheduledReports.updateReport** (idReport, idSite, description, period, hour, reportType, reportFormat, reports, parameters, idSegment = "") [ No example available ]

- **ScheduledReports.deleteReport** (idReport) [ No example available ]

- **ScheduledReports.getReports** (idSite = "", period = "", idReport = "", ifSuperUserReturnOnlySuperUserReports = "", idSegment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **ScheduledReports.generateReport** (idReport, date, language = "", outputType = "", period = "", reportFormat = "", parameters = "") [ No example available ]

- **ScheduledReports.sendReport** (idReport, period = "", date = "") [ No example available ]

[↑ Back to top](#)

## Module SegmentEditor

The SegmentEditor API lets you add, update, delete custom Segments, and list saved segments. @method static \Piwik\Plugins\SegmentEditor\API getInstance()

- **SegmentEditor.delete** (idSegment) [ No example available ]
- **SegmentEditor.update** (idSegment, name, definition, idSite = "", autoArchive = "", enabledAllUsers = "") [ No example available ]
- **SegmentEditor.add** (name, definition, idSite = "", autoArchive = "", enabledAllUsers = "") [ No example available ]
- **SegmentEditor.get** (idSegment) [ No example available ]
- **SegmentEditor.getAll** (idSite = "", returnOnlyAutoArchived = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

[↑ Back to top](#)

## Module SitesManager

The SitesManager API gives you full control on Websites in Piwik (create, update and delete), and many methods to retrieve websites based on various attributes. This API lets you create websites via "addSite", update existing websites via "updateSite" and delete websites via "deleteSite". When creating websites, it can be useful to access internal codes used by Piwik for currencies via "getCurrencyList", or timezones via "getTimezonesList". There are also many ways to request a list of websites: from the website ID via "getSiteFromId" or the site URL via "getSitesIdFromSiteUrl". Often, the most useful technique is to list all websites that are known to a current user, based on the token\_auth, via "getSitesWithAdminAccess", "getSitesWithViewAccess" or "getSitesWithAtLeastViewAccess" (which returns both). Some methods will affect all websites globally: "setGlobalExcludedIps" will set the list of IPs to be excluded on all websites, "setGlobalExcludedQueryParameters" will set the list of URL parameters to remove from URLs for all websites. The existing values can be fetched via "getExcludedIpsGlobal" and "getExcludedQueryParametersGlobal". See also the documentation about [Managing Websites](#) in Piwik.

@method static \Piwik\Plugins\SitesManager\API getInstance()

- **SitesManager.getJavascriptTag** (idSite, piwikUrl = "", mergeSubdomains = "", groupPageTitlesByDomain = "", mergeAliasUrls = "", visitorCustomVariables = "", pageCustomVariables = "", customCampaignNameQueryParam = "", customCampaignKeywordParam = "", doNotTrack = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getSitesFromGroup** (group) [ No example available ]
- **SitesManager.getSitesGroups** () [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getSiteFromId** (idSite) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getSiteUrlsFromId** (idSite) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getAllSites** () [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getAllSitesId** () [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getsitesIdWithVisits** (timestamp = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getsitesWithAdminAccess** () [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getsitesWithViewAccess** () [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]
- **SitesManager.getsitesWithAtLeastViewAccess** (limit = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **SitesManager.getsitesIdWithAdminAccess ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getsitesIdWithViewAccess ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getsitesIdWithAtLeastViewAccess ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getsitesIdFromSiteUrl (url)** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.addSite (siteName, urls, ecommerce = "", siteSearch = "", searchKeywordParameters = "", searchCategoryParameters = "", excludedIps = "", excludedQueryParameters = "", timezone = "", currency = "", group = "", startDate = "", excludedUserAgents = "", keepURLFragments = "", type = "")** [ No example available ]
- **SitesManager.deleteSite (idSite)** [ No example available ]
- **SitesManager.addSiteAliasUrls (idSite, urls)** [ No example available ]
- **SitesManager.getIpsForRange (ipRange)** [ No example available ]
- **SitesManager.setGlobalExcludedIps (excludedIps)** [ No example available ]
- **SitesManager.setGlobalSearchParameters (searchKeywordParameters, searchCategoryParameters)** [ No example available ]
- **SitesManager.getSearchKeywordParametersGlobal ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getSearchCategoryParametersGlobal ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getExcludedQueryParametersGlobal ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getExcludedUserAgentsGlobal ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.setGlobalExcludedUserAgents (excludedUserAgents)** [ No example available ]
- **SitesManager.isSiteSpecificUserAgentExcludeEnabled ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.setSiteSpecificUserAgentExcludeEnabled (enabled)** [ No example available ]
- **SitesManager.getKeepURLFragmentsGlobal ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.setKeepURLFragmentsGlobal (enabled)** [ No example available ]
- **SitesManager.setGlobalExcludedQueryParameters (excludedQueryParameters)** [ No example available ]
- **SitesManager.getExcludedIpsGlobal ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getDefaultCurrency ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.setDefaultCurrency (defaultCurrency)** [ No example available ]
- **SitesManager.getDefaultTimezone ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.setDefaultTimezone (defaultTimezone)** [ No example available ]
- **SitesManager.updateSite (idSite, siteName, urls = "", ecommerce = "", siteSearch = "", searchKeywordParameters = "", searchCategoryParameters = "", excludedIps = "", excludedQueryParameters = "", timezone = "", currency = "", group = "", startDate = "", excludedUserAgents = "", keepURLFragments = "", type = "")** [ No example available ]
- **SitesManager.getCurrencyList ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getCurrencySymbols ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getTimezonesList ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **SitesManager.getUniqueSiteTimezones ()** [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

- **SitesManager.getPatternMatchSites** (pattern) [ No example available ]

[↑ Back to top](#)

## Module Transitions

@method static \Piwik\Plugins\Transitions\API getInstance()

- **Transitions.getTransitionsForPageTitle** (pageTitle, idSite, period, date, segment = "", limitBeforeGrouping = "") [ No example available ]

- **Transitions.getTransitionsForPageUrl** (pageUrl, idSite, period, date, segment = "", limitBeforeGrouping = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **Transitions.getTransitionsForAction** (actionName, actionType, idSite, period, date, segment = "", limitBeforeGrouping = "", parts = 'all') [ No example available ]

- **Transitions.getTranslations** () [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

[↑ Back to top](#)

## Module UserCountry

The UserCountry API lets you access reports about your visitors' Countries and Continents. @method static \Piwik\Plugins\UserCountry\API getInstance()

- **UserCountry.getCountry** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **UserCountry.getContinent** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **UserCountry.getRegion** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **UserCountry.getCity** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **UserCountry.getLocationFromIP** (ip, provider = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]

- **UserCountry.getNumberOfDistinctCountries** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module UserSettings

The UserSettings API lets you access reports about your Visitors technical settings: browsers, browser types (rendering engine), operating systems, plugins supported in their browser, Screen resolution and Screen types (normal, widescreen, dual screen or mobile).  
@method static \Piwik\Plugins\UserSettings\API getInstance()

- **UserSettings.getResolution** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **UserSettings.getConfiguration** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

- **UserSettings.getOS** (idSite, period, date, segment = "", addShortLabel = '1') [ Example

in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getOSFamily** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getMobileVsDesktop** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getBrowserVersion** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getBrowser** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getBrowserType** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getWideScreen** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getPlugin** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

- **UserSettings.getLanguage** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module UsersManager

The UsersManager API lets you Manage Users and their permissions to access specific websites. You can create users via "addUser", update existing users via "updateUser" and delete users via "deleteUser". There are many ways to list users based on their login "getUser" and "getUsers", their email "getUserByEmail", or which users have permission (view or admin) to access the specified websites "getUsersWithSiteAccess". Existing Permissions are listed given a login via "getSitesAccessFromUser", or a website ID via "getUsersAccessFromSite", or you can list all users and websites for a given permission via "getUsersSitesFromAccess". Permissions are set and updated via the method "setUserAccess". See also the documentation about [Managing Users](#) in Piwik.

- **UsersManager.setUserPreference** (userLogin, preferenceName, preferenceValue) [ No example available ]

- **UsersManager.getUserPreference** (userLogin, preferenceName) [ No example available ]

- **UsersManager getUsers** (userLogins = "") [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getUsersLogin** () [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getUsersSitesFromAccess** (access) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getUsersAccessFromSite** (idSite) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getUsersWithSiteAccess** (idSite, access) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getSitesAccessFromUser** (userLogin) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getUser** (userLogin) [ Example in [XML](#), [Json](#), [Tsv](#) ([Excel](#)) ]

- **UsersManager.getUserByEmail** (userEmail) [ No example available ]

- **UsersManager.addUser** (userLogin, password, email, alias = "") [ No example available ]

- **UsersManager.updateUser** (userLogin, password = "", email = "", alias = "") [ No example available ]
- **UsersManager.deleteUser** (userLogin) [ No example available ]
- **UsersManager.userExists** (userLogin) [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) ]
- **UsersManager.userEmailExists** (userEmail) [ No example available ]
- **UsersManager.setUserAccess** (userLogin, access, idSites) [ No example available ]
- **UsersManager.getTokenAuth** (userLogin, md5Password) [ No example available ]

[↑ Back to top](#)

## Module VisitFrequency

VisitFrequency API lets you access a list of metrics related to Returning Visitors.  
@method static \Piwik\Plugins\VisitFrequency\API getInstance()

- **VisitFrequency.get** (idSite, period, date, segment = "", columns = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module VisitTime

VisitTime API lets you access reports by Hour (Server time), and by Hour Local Time of your visitors. @method static \Piwik\Plugins\VisitTime\API getInstance()

- **VisitTime.getVisitInformationPerLocalTime** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **VisitTime.getVisitInformationPerServerTime** (idSite, period, date, segment = "", hideFutureHoursWhenToday = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **VisitTime.getByDayOfWeek** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

## Module VisitorInterest

VisitorInterest API lets you access two Visitor Engagement reports: number of visits per number of pages, and number of visits per visit duration. @method static \Piwik\Plugins\VisitorInterest\API getInstance()

- **VisitorInterest.getNumberOfVisitsPerVisitDuration** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **VisitorInterest.getNumberOfVisitsPerPage** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **VisitorInterest.getNumberOfVisitsByDaysSinceLast** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]
- **VisitorInterest.getNumberOfVisitsByVisitCount** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#) , RSS of the last [10 days](#) ]

[↑ Back to top](#)

# Module VisitsSummary

VisitsSummary API lets you access the core web analytics metrics (visits, unique visitors, count of actions (page views & downloads & clicks on outlinks), time on site, bounces and converted visits. @method static \Piwik\Plugins\VisitsSummary\API getInstance()

- **VisitsSummary.get** (idSite, period, date, segment = "", columns = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getVisits** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getUniqueVisitors** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getActions** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getMaxActions** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getBounceCount** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getVisitsConverted** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getSumVisitsLength** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]
- **VisitsSummary.getSumVisitsLengthPretty** (idSite, period, date, segment = "") [ Example in [XML](#), [Json](#), [Tsv \(Excel\)](#), RSS of the last [10 days](#) ]

# Report Metadata API Methods

---

The standard Piwik Analytics APIs return raw data, for example: visits, page views, revenue, conversions.

The Metadata API is the service that calls other APIs, then add more metadata around it, for example, it will:

- return the report category, name, but also name of all the columns (in a specific language), or the translated date label (eg. "Thursday 10 February 2011")
- return the data units as well, including the % sign, money symbols like \$ or ₩, or time format "00:04:17"
- return the processed metrics (ratios, averages) that are not returned in the normal API response, for example: bounce rate and time on site. The Piwik Metadata API provides a simple entry point to get this additional information for most Piwik API functions returning analytics data.

# Fetch the Analytics Data and Metadata for a report

The method **API.getProcessedReport** can be called to request the full data set (metadata, column names, report data) of a given Piwik report. The response contains:

- 'metadata' - report name, category, list of metrics
- 'columns' - the list of translated column names in the report
- 'reportData' - the actual data: dimension and list of metrics
- 'reportMetadata' - (optional) contains the metadata for each row, for example: logo URLs, country codes, search engine URLs, etc. For example, if you want to display the top five countries that visitors come from you would build the API Request as follows:
- User countries report: &apiModule=UserCountry&apiAction=getCountry
- Truncated to 5 rows: &filter\_truncate=5
- Labels can be translated into a specific language. As with other API calls, you can use the parameter &language=xx (replacing xx with the translation code). The URL would be[http://demo.piwik.org/?module=API&method=API.getProcessedReport&idSite=3&date=yesterday&period=day&apiModule=UserCountry&apiAction=getCountry&language=en&format=xml&token\\_auth=anonymous&filter\\_truncate=5](http://demo.piwik.org/?module=API&method=API.getProcessedReport&idSite=3&date=yesterday&period=day&apiModule=UserCountry&apiAction=getCountry&language=en&format=xml&token_auth=anonymous&filter_truncate=5)

The returned XML is:

```
<?xml version="1.0" encoding="utf-8" ?>
<result>
    <website>virtual-drums.com</website>
    <prettyDate>Thursday 19 December 2013</prettyDate>
    <metadata>
        <category>Visitors</category>
        <name>Country</name>
        <module>UserCountry</module>
        <action>getCountry</action>
        <dimension>Country</dimension>
        <metrics>
            <nb_visits>Visits</nb_visits>
```

```
<nb_uniq_visitors>Unique visitors</nb_uniq_visitors>

<nb_actions>Actions</nb_actions>

</metrics>

<processedMetrics>

<nb_actions_per_visit>Actions per Visit</nb_actions_per_visit>

<avg_time_on_site>Avg. Time on Website</avg_time_on_site>

<bounce_rate>Bounce Rate</bounce_rate>

</processedMetrics>

<metricsDocumentation>

<nb_visits>If a visitor comes to your website for the first time or if he visits a page more than 30 minutes after his last page view, this will be recorded as a new visit.</nb_visits>

<nb_uniq_visitors>The number of unduplicated visitors coming to your website. Every user is only counted once, even if he visits the website multiple times a day.</nb_uniq_visitors>

<nb_actions>The number of actions performed by your visitors. Actions can be page views, internal site searches, downloads or outlinks.</nb_actions>

<nb_actions_per_visit>The average number of actions (page views, site searches, downloads or outlinks) that were performed during the visits.</nb_actions_per_visit>

<avg_time_on_site>The average duration of a visit.</avg_time_on_site>

<bounce_rate>The percentage of visits that only had a single pageview. This means, that the visitor left the website directly from the entrance page.</bounce_rate>

<conversion_rate>The percentage of visits that triggered a goal conversion.</conversion_rate>

<avg_time_on_page>The average amount of time visitors spent on this page (only the page, not the entire website).</avg_time_on_page>
```

```
        <nb_hits>The number of times this page was  
visited.</nb_hits>  
        <exit_rate>The percentage of visits that l  
eft the website after viewing this page.</exit_rate>  
    </metricsDocumentation>  
    <metricsGoal>  
        <nb_conversions>Conversions</nb_conversion  
s>  
        <revenue>Revenue</revenue>  
    </metricsGoal>  
    <processedMetricsGoal>  
        <revenue_per_visit>Revenue per Visit</reve  
nue_per_visit>  
    </processedMetricsGoal>  
    <imageGraphUrl>index.php?module=API&method=Im  
ageGraph.get&idSite=3&apiModule=UserCountry&apiAct  
ion=getCountry&token_auth=anonymous&period=day&dat  
e=yesterday</imageGraphUrl>  
    <imageGraphEvolutionUrl>index.php?module=API&  
method=ImageGraph.get&idSite=3&apiModule=UserCountry&  
apiAction=getCountry&token_auth=anonymous&period=da  
y&date=2013-11-20,2013-12-19</imageGraphEvolutionUrl>  
    <uniqueId>UserCountry_getCountry</uniqueId>  
</metadata>  
    <columns>  
        <label>Country</label>  
        <nb_visits>Visits</nb_visits>  
        <nb_uniq_visitors>Unique visitors</nb_uniq_visito  
rs>  
        <nb_actions>Actions</nb_actions>  
        <nb_actions_per_visit>Actions per Visit</nb_actio  
ns_per_visit>  
        <avg_time_on_site>Avg. Time on Website</avg_time_  
on_site>
```

```
<bounce_rate>Bounce Rate</bounce_rate>

<revenue>Revenue</revenue>

</columns>

<reportData>

    <row>

        <label>United States</label>

        <nb_uniq_visitors>20</nb_uniq_visitors>

        <nb_visits>20</nb_visits>

        <nb_actions>20</nb_actions>

        <nb_actions_per_visit>1</nb_actions_per_visit>

        <avg_time_on_site>00:00:00</avg_time_on_site>

        <bounce_rate>100%</bounce_rate>

        <revenue>$ 0</revenue>

    </row>

    <row>

        <label>Indonesia</label>

        <nb_uniq_visitors>17</nb_uniq_visitors>

        <nb_visits>17</nb_visits>

        <nb_actions>18</nb_actions>

        <nb_actions_per_visit>1.06</nb_actions_per_visit>

        <avg_time_on_site>00:00:02</avg_time_on_site>

        <bounce_rate>94.12%</bounce_rate>

        <revenue>$ 0</revenue>

    </row>

    <row>

        <label>Philippines</label>

        <nb_uniq_visitors>9</nb_uniq_visitors>
```

```
<nb_visits>9</nb_visits>
<nb_actions>10</nb_actions>
<nb_actions_per_visit>1.11</nb_actions_per_visit>
<avg_time_on_site>00:00:02</avg_time_on_site>
<bounce_rate>88.89%</bounce_rate>
<revenue>$ 0</revenue>
</row>
<row>
<label>United Kingdom</label>
<nb_uniq_visitors>7</nb_uniq_visitors>
<nb_visits>7</nb_visits>
<nb_actions>10</nb_actions>
<nb_actions_per_visit>1.43</nb_actions_per_visit>
<avg_time_on_site>00:01:01</avg_time_on_site>
<bounce_rate>85.71%</bounce_rate>
<revenue>$ 0</revenue>
</row>
<row>
<label>France</label>
<nb_uniq_visitors>3</nb_uniq_visitors>
<nb_visits>3</nb_visits>
<nb_actions>4</nb_actions>
<nb_actions_per_visit>1.33</nb_actions_per_visit>
<avg_time_on_site>00:00:02</avg_time_on_site>
<bounce_rate>66.67%</bounce_rate>
<revenue>$ 0</revenue>
```

```
</row>

<row>
    <label>Others</label>
    <nb_uniq_visitors>26</nb_uniq_visitors>
    <nb_visits>26</nb_visits>
    <nb_actions>33</nb_actions>
    <nb_actions_per_visit>1.27</nb_actions_per_visit>
    <avg_time_on_site>00:00:04</avg_time_on_site>
    <bounce_rate>76.92%</bounce_rate>
    <revenue>$ 0</revenue>
</row>
</reportData>
<reportMetadata>
    <row>
        <code>us</code>
        <logo>plugins/UserCountry/images/flags/us.png</logo>
        <logoWidth>16</logoWidth>
        <logoHeight>11</logoHeight>
    </row>
    <row>
        <code>id</code>
        <logo>plugins/UserCountry/images/flags/id.png</logo>
        <logoWidth>16</logoWidth>
        <logoHeight>11</logoHeight>
    </row>
    <row>
        <code>ph</code>
```

```
        <logo>plugins/UserCountry/images/flags/ph.
png</logo>
            <logoWidth>16</logoWidth>
            <logoHeight>11</logoHeight>
        </row>
        <row>
            <code>gb</code>
            <logo>plugins/UserCountry/images/flags/gb.
png</logo>
            <logoWidth>16</logoWidth>
            <logoHeight>11</logoHeight>
        </row>
        <row>
            <code>fr</code>
            <logo>plugins/UserCountry/images/flags/fr.
png</logo>
            <logoWidth>16</logoWidth>
            <logoHeight>11</logoHeight>
        </row>
        <row>
            <logoWidth>16</logoWidth>
            <logoHeight>11</logoHeight>
        </row>
    </reportMetadata>
    <reportTotal>
        <nb_visits>82</nb_visits>
        <nb_uniq_visitors>82</nb_uniq_visitors>
        <nb_actions>95</nb_actions>
        <nb_visitsConverted>0</nb_visitsConverted>
        <bounce_count>72</bounce_count>
    </reportTotal>
```

```
<timerMillis>55</timerMillis>  
</result>
```

## List and Definition of 'metadata' Response Attributes

- **category** - category under which the report appears
- **name** - the report name
- **module** and 'action' - used to build the standard API request to fetch the data for this report, ?module=API&method=\$module. \$action (replace \$module by the 'module' attribute, replace \$action by the 'action' attribute)
- **metrics** - list of basic metrics in the report
- **processedMetrics** - list of processed metrics in the report. Processed metrics are usually ratios (conversion rates, average actions per visit, etc.). Processed metrics don't appear in standard non-metadata API responses.
- **metricsGoal** and 'processedMetricsGoal' - list of goal metrics available for this report.
- **uniqueId** - the report unique ID.
- **constantRowsCount** - if set to 1, this means that the report always has the same number of rows. For example, "visits per hour" always has 24 hours, "visits per number of pages" has 10 rows. This attribute is set only when there is a 'dimension' attribute. If this attribute is not set, it means that the returned data set could have 0 row, 1 row, or N rows.

## Listing all the Metadata API Functions

The API method **API.getReportMetadata** can be called to request the full list of API functions returning web analytics reports - [see the example output on the Piwik demo](#).

There are two types of reports in Piwik, and each have a slightly different format.

- **Simple metrics reports**

Simple Metrics reports simply contain a list of metrics and their values. For example, VisitsSummary.get returns the main metrics (visits, pages, unique visitors) for the specified website ([example URL](#)).

```
<?xml version="1.0" encoding="utf-8" ?>  
<result>  
  <row>  
    <category>Visits Summary</category>
```

```
<name>Visits Summary</name>
<module>VisitsSummary</module>
<action>get</action>
<metrics>
    <nb_uniq_visitors>Unique visitors</nb_uniq_visitors>
    <nb_visits>Visits</nb_visits>
    <nb_actions>Actions</nb_actions>
    <nb_actions_per_visit>Actions per Visit</nb_actions_per_visit>
    <bounce_rate>Bounce Rate</bounce_rate>
    <avg_time_on_site>Avg. Visit Duration (in seconds)</avg_time_on_site>
    <max_actions>Maximum actions in one visit</max_actions>
</metrics>
<metricsDocumentation>
    <nb_visits>If a visitor comes to your website for the first time or if he visits a page more than 30 minutes after his last page view, this will be recorded as a new visit.</nb_visits>
    <nb_uniq_visitors>The number of unduplicated visitors coming to your website. Every user is only counted once, even if he visits the website multiple times a day.</nb_uniq_visitors>
    <nb_actions>The number of actions performed by your visitors. Actions can be page views, internal site searches, downloads or outlinks.</nb_actions>
    <nb_actions_per_visit>The average number of actions (page views, site searches, downloads or outlinks) that were performed during the visits.</nb_actions_per_visit>
    <avg_time_on_site>The average duration of a visit.</avg_time_on_site>
    <bounce_rate>The percentage of visits that only had a single pageview. This means, that the visitor
```

```

left the website directly from the entrance page.</bounce_
rate>

<conversion_rate>The percentage of visits
that triggered a goal conversion.</conversion_rate>

<avg_time_on_page>The average amount of ti
me visitors spent on this page (only the page, not the ent
ire website).</avg_time_on_page>

<nb_hits>The number of times this page was
visited.</nb_hits>

<exit_rate>The percentage of visits that l
eft the website after viewing this page.</exit_rate>

</metricsDocumentation>

<imageGraphUrl>index.php?module=API&method=Im
ageGraph.get&idSite=3&apiModule=VisitsSummary&
amp;apiAction=get&token_auth=anonymous&period=day&
amp;date=2013-11-21,2013-12-20</imageGraphUrl>

<imageGraphEvolutionUrl>index.php?module=API&
method=ImageGraph.get&idSite=3&apiModule=VisitsSu
mmary&apiAction=get&token_auth=anonymous&peri
od=day&date=2013-11-21,2013-12-20</imageGraphEvolutio
nUrl>

<uniqueId>VisitsSummary_get</uniqueId>

</row>

</result>
```

- **Reports with dimensions**

Most reports, however, will have a 'dimension' entry in the returned array. Reports with dimensions will display a list of metrics for each 'dimension'. For example, the list of visits, pages, time on site will be output for each keyword.

Example of a report with dimensions metadata ([example URL](#)):

```
<?xml version="1.0" encoding="utf-8" ?>

<result>0</result>
```

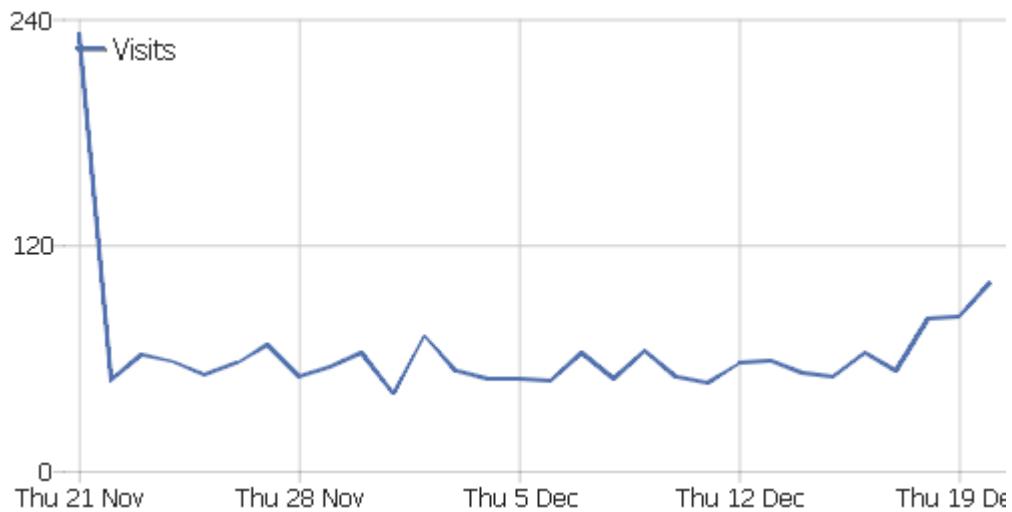
## Static Image Graphs

In the metadata output, the field <imageGraphUrl> is a URL that will generate a static PNG graph plotting data for the requested report. Static PNG graphs are used, for

example, in the [Piwik mobile app](#) and in [email reports](#). These static image graphs can also be used in any custom dashboard, web page, monitoring page, email, etc. As opposed to the [Piwik Widgets](#), static image graphs do not require JavaScript or HTML, since the URL returns a PNG image.

In the following examples, to see the URL used to generate the image, right-click on the image and select "view image" to see the full URL.

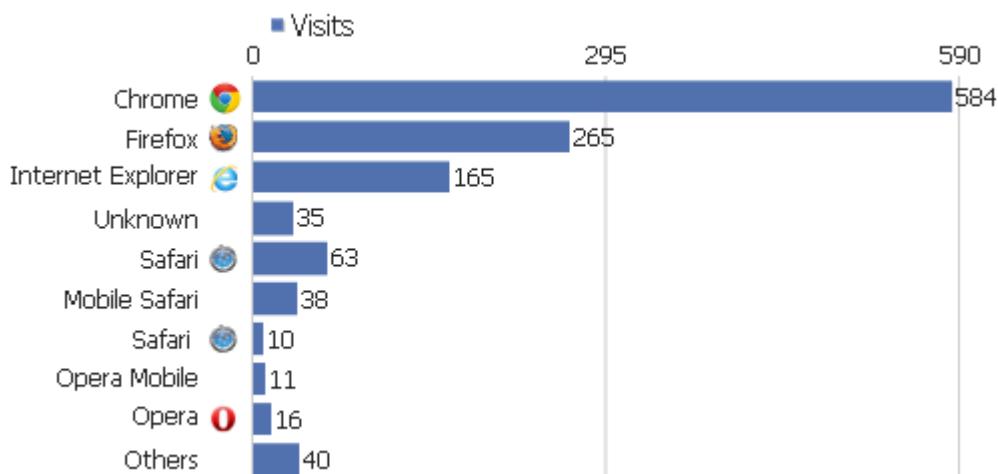
- Example: Graph Plotting Visits over the Last 30 Days



URL =

`index.php?module=API&method=ImageGraph.get&idSite=3&apiModule=VisitsSummary&apiAction=get&token_auth=anonymous&graphType=evolution&period=day&date=previous30&width=500&height=250`

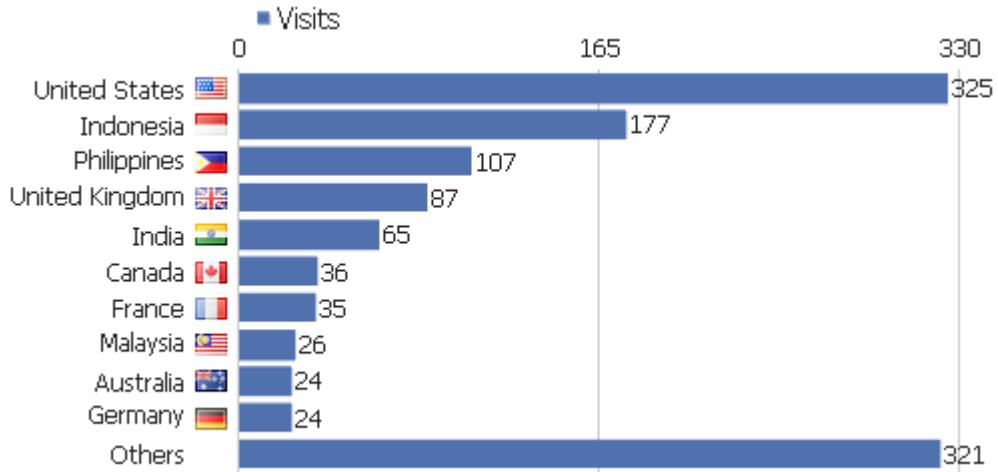
- Example: Horizontal Bar Graph Plotting Browsers for the Current Month



*URL =*

*index.php?module=API&method=ImageGraph.get&idSite=3&apiModule=UserSettings&apiAction=getBrowser&token\_auth=anonymous&graphType=horizontalBar&period=month&date=today&width=500&height=250*

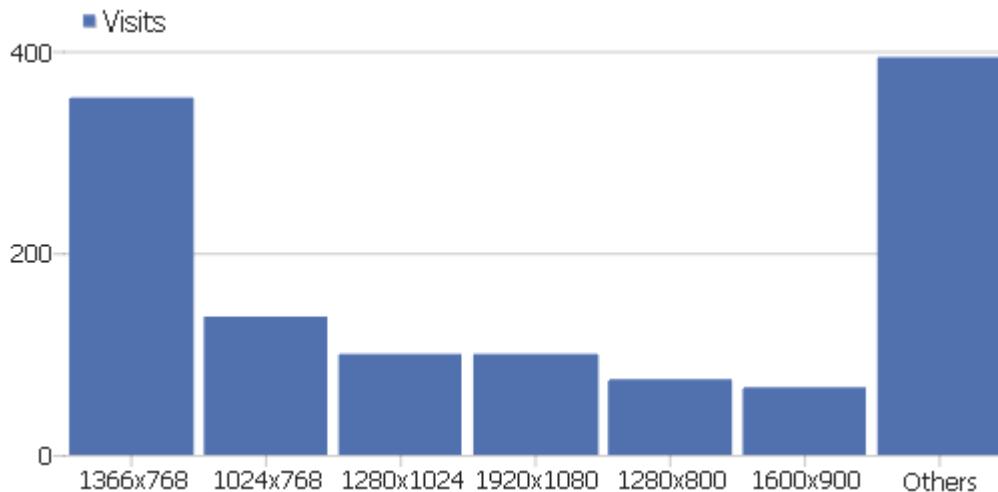
- Example: Horizontal Bar Graph Plotting Countries for the Current Week



*URL =*

*index.php?module=API&method=ImageGraph.get&idSite=3&apiModule=UserSettings&apiAction=getResolution&token\_auth=anonymous&graphType=verticalBar&period=month&date=today&width=500&height=250*

- Example: Graph Plotting User Screen Resolutions for the Current Month



*URL =*

*index.php?module=API&method=ImageGraph.get&idSite=3&apiModule=UserSettings&apiAction=getResolution&token\_auth=anonymous&graphType=verticalBar&period=month&date=today&width=500&height=250*

- Example: Pie Chart with Custom Colors

*URL =*

```
index.php?module=API&method=ImageGraph.get&idSite=7&apiModule=UserSettings&a  
piAction=getOS&token_auth=anonymous&graphType=pie&period=month&date=today&  
width=500&height=250&column=nb_visits&colors=FFFF00,00FF00,FF0000,0000FF,555  
555,C3590D
```

The static Graphs API requires the standard Piwik parameters (idSite, date, period, etc.) but also accepts the following parameters:

- **graphType** - defines the type of graph to draw. Accepted values are: '**evolution**' (line graph), '**horizontalBar**' (horizontal bar graph), '**verticalBar**' (vertical bar graph) and '**pie**' (2D Pie chart)
- **width** and **height** - define the width and height in pixels of the generated image
- **column** - by default, the graph will plot the number of visits (nb\_visits). You can specify another metric such as: nb\_actions, nb\_visitsConverted, etc.
- **colors** - you can specify a comma separated list of hexadecimal colors to use in the graph instead of the default colors, eg. &colors=FFFF00,00FF00,FF0000
- **aliasedGraph** - by default, graphs are "smooth" (anti-aliased). If you are generating hundreds of graphs and are concerned with performance, you can set aliasedGraph=0. This will disable anti-aliasing and graphs will be generated faster, but won't look so pretty.

## Segmentation in the API

---

Learn how to use the powerful segmentation feature available in Piwik. This page explains how to build the 'segment' parameter in your API requests. Segmentation makes it easy to request any Piwik report for a subset of your audience.

This page explains how to build and use the 'segment' API URL parameter, and you will find the list of all the supported visitor segments (country, entry page, keyword, returning visitor, etc.).

### Segment Parameter in the API URL Request

Segmentation can be applied to most API functions. The **segment** parameter contains the definition of the segment you wish to filter your reports to.

For example, you can request the "Best Keywords" report processed for all visits where "Country is Germany AND Browser is Firefox" (), by doing the following request:

```

http://piwik.example.org/index.php
?token_auth=yourTokenHere
&format=xml
&date=2011-01-11
&period=week
&idSite=1
&module=API&method=Referers.getKeywords
&segment=browserName==FF;country==DE</div>

```

Example URL of top countries used by visits landing on the page: [virtual-drums.com/:demo.piwik.org/?module=API&method=UserCountry.getCountry&idSite=3&date=yesterday&period=day&format=xml&filter\\_truncate=5&language=en&segment=entryPageUrl==http%3A%2F%2Fwww.virtual-drums.com%2F](http://virtual-drums.com/:demo.piwik.org/?module=API&method=UserCountry.getCountry&idSite=3&date=yesterday&period=day&format=xml&filter_truncate=5&language=en&segment=entryPageUrl==http%3A%2F%2Fwww.virtual-drums.com%2F)

Let's take a look at the segment string.

## Segment operators

<code>==</code> Equals	<code>&amp;segment=country==IN</code>	Return results where the country is India
<code>!=</code> Does not equal	<code>&amp;segment=actions!=1</code>	Return results where the number of actions (page views, downloads, etc.) is not 1
<code>&lt;=</code> Less than or equal to	<code>&amp;segment=actions&lt;=4</code>	Return results where the number of actions (page views, downloads, etc.) is 4 or less
<code>&lt;</code> Less than	<code>&amp;segment=visitServerHour&lt;12</code>	Return results where the Server time (hour) is before midday.
<code>&gt;=</code> Greater than or equal to	<code>&amp;segment=visitDuration&gt;=600</code>	Return results where visitors spent 10 minutes or more on the website.
<code>&gt;</code> Greater than	<code>&amp;segment=daysSinceLastVisit&gt;1</code>	Return results where visitors are coming back to the website 2 days or more after their previous visit.
<code>=@</code> Contains	<code>&amp;segment=referrerName=@piwik</code>	Return results where the Referrer name (website domain or search engine name) contains the word "piwik".
<code>!@</code> Does not contain	<code>&amp;segment=referrerKeyword!@yourBrand</code>	Return results where the keyword used to access the website does not contain word "yourBrand".

# Combine Segments with AND and OR expressions

You can combine several segments together with AND and OR logic.

**OR** operator is the , (comma) character.

Examples

```
&segment=country==US, country==DE
```

Country is either (United States OR Germany)

**AND** operator is the ; (semi-colon) character.

Examples

```
&segment=visitorType==returning; country==FR
```

Returning visitors AND Country is France

```
&segment=referrerType==search; referrerKeyword!=Piwik
```

Visitors from Search engines AND Keyword is not Piwik

Note that if you combine OR and AND operators, the OR operator will take precedence.

For example, the following query

```
&segment=referrerType==search; referrerKeyword==Piwik, referrerK  
eyword==analytics
```

will select "Visitors from Search engines AND (Keyword is Piwik OR Keyword is analytics)"

## List of segments

### Dimensions

#### Visit Location

city	City Example values: <code>Sydney, Sao Paolo, Rome, etc.</code>
continentCode	Continent Example values: <code>eur, asi, amc, amn, ams, afr, ant, oce</code>
countryCode	Country Example values: <code>de, us, fr, in, es, etc.</code>
latitude	Latitude Example values: <code>-33.578, 40.830, etc.</code> You can select visitors within a lat/long range using <code>&amp;segment=lat&gt;X;lat&lt;Y;long&gt;M;long</code>

	<N.
longitude	Longitude Example values: <b>-70.664, 14.326, etc.</b>
provider	Provider Example values: <b>comcast.net, proxad.net, etc.</b>
regionCode	Region Example values: <b>01 02, OR, P8, etc.</b> eg. region=A1;country=fr
<b>Visit</b>	
browserCode	Browser Example values: <b>FF, IE, CH, SF, OP, etc.</b>
browserVersion	Browser version Example values: <b>1.0, 8.0, etc.</b>
deviceType	Device type Example values: <b>desktop, smartphone, tablet, feature phone, console, tv, car browser</b>
visitLocalHour	Local time Example values: <b>0, 1, 2, 3, ..., 20, 21, 22, 23</b>
operatingSystemCode	Operating system Example values: <b>WXP, WI7, MAC, LIN, AND, IPD, etc.</b>
resolution	Resolution Example values: <b>1280x1024, 800x600, etc.</b>
visitServerHour	Server time Example values: <b>0, 1, 2, 3, ..., 20, 21, 22, 23</b>
visitEcommerceStatus	Visit Ecommerce status at the end of the visit Example values: <b>none, ordered, abandonedCart, orderedThenAbandonedCart.</b> For example, to select all visits that have made an Ecommerce order, the API request would contain " <b>&amp;segment=visitEcommerceStatus==ordered,visitEcommerceStatus==orderedThenAbandonedCart</b> "

	Visit ID
visitId	<p>Note: This segment can only be used by an Admin user</p> <p>Example values:</p> <p>Any integer.</p>
visitConvertedGoalId	<p>Visit converted a specific Goal Id</p> <p>Example values:</p> <p>1, 2, 3, etc.</p>
visitConverted	<p>Visit converted at least one Goal</p> <p>Example values:</p> <p>0, 1</p>
visitorId	<p>Visitor ID</p> <p>Note: This segment can only be used by an Admin user</p> <p>Example values:</p> <p>34c31e04394bdc63 - any 16 Hexadecimal chars ID, which can be fetched using the Tracking API function <b>getVisitorId()</b></p>
visitorType	<p>Visitor type</p> <p>Example values:</p> <p>new, returning, returningCustomer. For example, to select all visitors who have returned to the website, including those who have bought something in their previous visits, the API request would contain "&amp;segment=visitorType==returning,visitorType==returningCustomer"</p>
<b>Referrers</b>	
referrerKeyword	<p>Keyword</p> <p>Example values:</p> <p>Encoded%20Keyword, keyword</p>
referrerName	<p>Referrer Name</p> <p>Example values:</p> <p>twitter.com, www.facebook.com, Bing, Google, Yahoo, CampaignName</p>
referrerType	<p>Referrer Type</p> <p>Example values:</p> <p>direct, search, website, campaign</p>
referrerUrl	<p>Referrer URL</p> <p>Example values:</p> <p>http%3A%2F%2Fwww.example.org%2Freferer-page.htm</p>
<b>Events</b>	
eventAction	Event Action

eventCategory	Event Category
eventName	Event Name
<b>Custom Variables</b>	
customVariableName1	Custom Variable name 1 (scope visit)
customVariableName2	Custom Variable name 2 (scope visit)
customVariablePageName1	Custom Variable name 1 (scope page)
customVariablePageValue1	Custom Variable value 1 (scope page)
customVariableValue1	Custom Variable value 1 (scope visit)
customVariableValue2	Custom Variable value 2 (scope visit)

There are 5 custom variables available, so you can segment across any segment name and value range.

For example,

---

`customVariableName1==Type;customVariableValue1==Customer`

Returns all visitors that have the Custom Variable "Type" set to "Customer".  
 Custom Variables of scope "page" can be queried separately. For example, to query the Custom Variable of scope "page",  
 stored in index 1, you would use the segment  
`customVariablePageName1==ArticleLanguage;customVariablePageValue1==FR`

---

### Actions

entryPageUrl	Entry Page URL
entryPageTitle	Entry Page title
exitPageTitle	Exit Page Title
exitPageUrl	Exit Page URL
siteSearchKeyword	Keyword (Site Search)
pageTitle	Page Name
	Page URL
	Example values:
pageUrl	All these segments must be URL encoded, for example: <code>http%3A%2F%2Fexample.com%2Fpath%2Fpage%3Fquery</code>

### Metrics

#### Visit

daysSinceFirstVisit	Days since first visit
daysSinceLastEcommerceOrder	Days since last Ecommerce order
daysSinceLastVisit	Days since last visit
actions	Number of Actions
	Number of Events
events	Example values: To select all visits who triggered an Ev

	<code>ent, use: &amp;segment=events&gt;0</code>
searches	Number of Internal Searches Example values: <code>To select all visits who used internal Site Search, use: &amp;segment=searches&gt;0</code>
visitCount	Number of visits
visitDuration	Visit Duration (in seconds)
visitIp	Visitor IP Note: This segment can only be used by an Admin user Example values: <code>13.54.122.1.</code> Select IP ranges with notation: <code>visitIp&gt;13.54.122.0;visitIp&lt;13.54.122.255</code>

## Segment where value is empty / is not empty

You may sometimes want to segment your analytics reports, for all visitors where a given dimension is empty (a value was not set). This is similar to the SQL "is null" clause. To do so, you can leave the value blank after the operator

`==`

in the segment string. For example, to select all visitors that did not have any referrer keyword set, you can write:

`referrerKeyword==`

You can combine it with other segments, for example to select all visitors that come from India and did not have any keyword set:

`referrerKeyword==;countryCode==in`

Similarly, you can segment your traffic to select visitors where a particular segment is not empty (a value was set). This is similar to the SQL "is not null" clause. To do so, you can leave the value blank after the operator

`!=`

in the segment string. For example to select all visitors that come from India and have a City set, you can write:

`city!=;countryCode==in`

*Note: Leaving an empty value is supported for the operators == and !=*