

Technical Note:

Working with the Panoramio API

NIH Proposal for Food Environment

Project conducted by:

San Diego State University

5500 Campanile Drive

San Diego, CA 92182-4493

Report prepared by:

Ying (Joey) Lee & Hao Zhang

Department of Geography, San Diego State University

February 2015

***Abstract***

This report introduces the Panoromia API, which allows developer to connect to the Panoramio database for retrieval of users and photos’ information. This report covers details about using the API, from getting started with applying the API keys to using the API method and retrieving data. There are many methods can be used from an endpoint application, however, this report will only focus on how to search and obtain information about requesting photo and its id. The data retrieved from Panoramio with the API has potential to be used for mapping and analyzing human dynamic.

This report covers the following items:

# Obtaining the JavaScript library

# Class Photo Request

# Class Widget

***Working with the Panoromia API***

Introduction

Panoramio is a geolocation-oriented photo sharing mashup owned by Google. Accepted photos uploaded to the site can be accessed as a layer in Google Earth and Google Maps, with new photos being added at the end of every month.

In this report, we will focus on the requesting photos and photos’ information. To demonstrate the use of API, we will use ***Javascript***as the programming language. This report uses the ***Panoramio Widget API Reference*** created by Panoramio as a reference.

Obtaining the JavaScript library

To use the JavaScript Widget API in your web page, you need to load the following URLhttp://www.panoramio.com/wapi/wapi.js?v=1to obtain a version localized to the language set in the user's browser configuration, orhttp://www.panoramio.com/wapi/wapi.js?v=1&hl=deto obtain a version localized to a particular language, German in the example (German's language code is "de"; you can use other language codes). This needs to be loaded in the *head*section of your page, so your HTML code will end up looking similar to this:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr">

<head>

<title>My page</title>

<meta name="description" content="This is my page." />

<link rel="shortcut icon" type="image/x-icon" href="/img/favicon.ico" />

<link rel="stylesheet" type="text/css" href="/css/page.css" />

<script type="text/javascript" src="http://www.panoramio.com/wapi/wapi.js?v=1&amp;hl=fr"></script>

</head>

<body>

<h1>My page</h1>

...

A request is a JavaScript object that specifies which set of photos should be displayed by a widget. You can reuse the same request in multiple widgets. You can specify a request with a JavaScript object literal. To build a request in this way, use code like this in JavaScript:

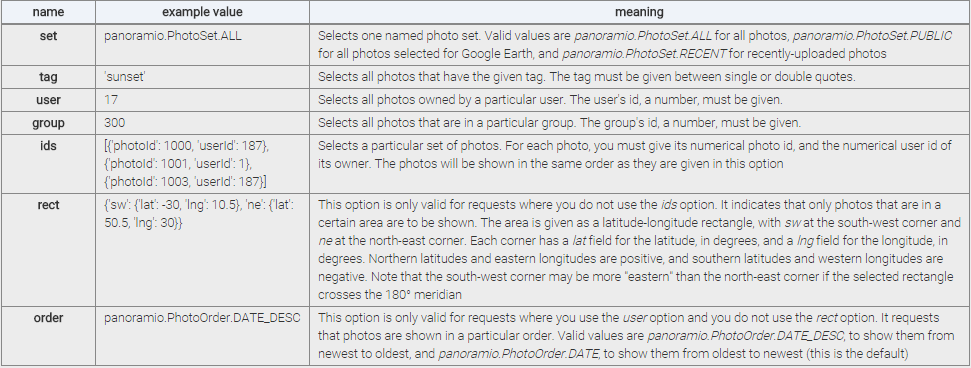
var myRequest = {

'tag': 'sunset',

'rect': {'sw': {'lat': -30, 'lng': 10.5}, 'ne': {'lat': 50.5, 'lng': 30}}

};

These are the valid fields for a request object:



Some requests contain an element of randomness, they may return different results if you run them twice. This can be problematic if you have two or more widgets that are supposed to show the same results, and there is some linkage between the widgets —for example clicking on one photo in a photo thumbnail list displays that photo in a larger format in another widget. To avoid this you can use a *panoramio.PhotoRequest* object built with a JavaScript constructor. If you pass the same *panoramio.PhotoRequest* object to two or more widgets, they will share the results of the request and the problem described will not happen. You can create a *panoramio.PhotoRequest* object in this way by passing it request options, for example with

var myRequest = new panoramio.PhotoRequest({

'tag': 'sunset',

'rect': {'sw': {'lat': -30, 'lng': 10.5}, 'ne': {'lat': 50.5, 'lng': 30}}

};

Class PhotoRequest

A PhotoRequest defines a "result set". Given constraints on what photos we want to seach for (for example, all photos in group "Snow") the result set is the ordered list of photos that satisfy the constraint. Throughout the WAPI, photos obtained as a result of a request are referenced through their position in the result set, also called their "index". That is, photo at position 5 in the result set for the query "popular photos" will be the 5th most popular photo.

The following table shows the class photo request opinion which include the field name and description.

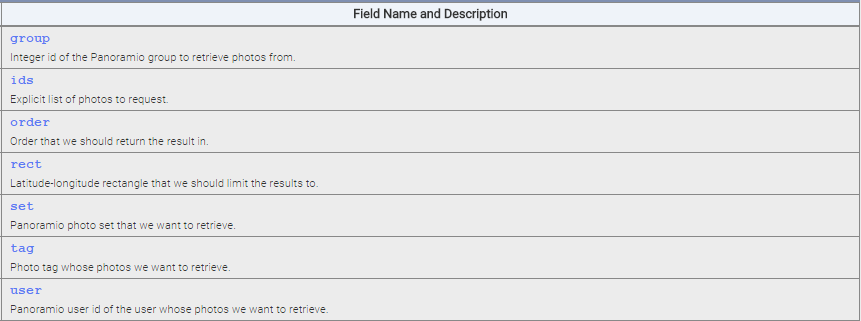


Table 1. Field Name and Description

The following context describe the detail of retrieving photo information from Panoramio.

{(number|undefined)} **group**

Integer id of the Panoramio group to retrieve photos from.

{(Array.<Object>|undefined)} **ids**

Explicit list of photos to request. The items of the array have to have two fields, the "userId" field has to contain the Panoramio id, an integer, of the photo owner and the "photoId" field has to contain the Panoramio photo id, as an integer.  
For example,

'ids': [{'userId': 187, 'photoId': 1000},

{'userId': 1, 'photoId': 31444256},

{'userId', 1, 'photoId': 30880260}}]

{([panoramio.PhotoOrder](http://www.panoramio.com/api/widget/reference.html" \l "PhotoOrder)|undefined)} **order**

Order that we should return the result in.   
This option is not valid if the 'rect' field is used, if the 'ids' field is used, if the 'set' field is used, or if the 'tag' field is used but not the 'user' field. It defaults to DATE, which means from oldest to newest.

{({sw:{lat:number, lng:number}, ne:{lat:number, lng:number}}|undefined)} **rect**

Latitude-longitude rectangle that we should limit the results to.  
The 'ne' and 'sw' fields are the north-east and south-west corners of the rectangle, respectively, and each point has a 'lat' and 'lng' field with the latitude in degrees north and longitude in degrees east respectively (so negative latitudes are in the southern hemisphere, and negative longitudes are west of the Greenwith meridian). If a rectangle crosses the 180 degree meridian, the longitude of its north-east corner will be smaller than that of its south-west corner. This option is not valid if the 'ids' field is used, if the 'order' field is used, or if the 'user' and 'tag' are used at the same time, or if the 'user' and 'group' are used at the same time.

{([panoramio.PhotoSet](http://www.panoramio.com/api/widget/reference.html" \l "PhotoSet)|undefined)} **set**

Panoramio photo set that we want to retrieve.

{(string|undefined)} **tag**

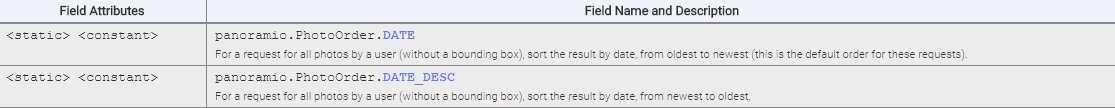
Photo tag whose photos we want to retrieve

{(number|undefined)} **user**

Panoramio user id of the user whose photos we want to retrieve.

Class PhotoOder

The following context describe the detail of information for photo order from Panoramio



Class Widget

Parent class for all widgets.  
Currently, this class has two purposes:

Defines a parent type that can be used in type signatures when any widget is needed, and

Defines code that is shared by the PhotoWidget and the PhotoListWidget.

Since currently these are the only two widgets, this is ok. Once we add more widgets, it may make sense to split this into two classes, a Widget class that acts as the hierarchy root, and a PhotoWidgetParent class that provides shared code by those two widgets.

The detail of this method is described below:

enableNextArrow(enable)

Control the visibility and clicking behaviour of the "next" arrow.  
This is meant to be called manually by the user.

Parameters:

{boolean} enable

Whether to enable (if true) or disable (if false) the "next" or right/down-pointing arrow.

enablePreviousArrow(enable)

Control the visibility and clicking behaviour of the "previous" arrow.  
This is meant to be called manually by the user.

Parameters:

{boolean} enable

Whether to enable (if true) or disable (if false) the "previous" or left/up-pointing arrow.

{?boolean} getAtEnd()

Returns whether we are at the end of the result set.

Returns:

{?boolean} Whether we are at the end of the result set, and therefore cannot move forwards. Returns 'null' if a request has not yet been defined, or we have not moved into it.

{?boolean} getAtStart()

Returns whether we are at the beginning of the result set.

Returns:

{?boolean} Whether we are at the beginning of the result set, and therefore cannot move backwards. Returns 'null' if a request has not yet been defined, or we have not moved into it.

To achieve the list of photo can be done through the following method

{Array.<?panoramio.Photo>} **getPhotos**()

Gets the currently displayed photos.

Returns:

{Array.<?panoramio.Photo>} For each slot in the widget, the photo displayed in that slot, or 'null' if no photo is shown at that slot (for example because the widget is not fully initialized, or because the slot's position in the result set is out of range). Note that only after the panoramio.events.EventType.PHOTO\_CHANGED event has been raised will this return the new photos.

{?number} **getPosition**()

Gets the current position in the result set.

Returns:

{?number} Position of the widget in the photo set. For this multi-photo widget, the position is that of the photo shown in the first slot (at the upper-left corner) except that it is possible for this to be a negative number if we start showing results at a later slot and the first few slots are blank. It will be 'null' if the widget is not fully initialized. Note that only after the panoramio.events.EventType.PHOTO\_CHANGED event has been raised will this return the new position.

**setPosition**(index)

Moves to a particular position in the result set.  
If that position is close enough to the current one, a sliding animation will be shown. If no request is defined, a blank photo area will be shown.  
This method needs to be called for the widget to start displaying anything. This lets the user install handlers for the PHOTO\_CHANGED event after the widget is constructed but before the event is raised.  
Raises a panoramio.events.EventType.PHOTO\_CHANGED event after the new photos are displayed (or after it has been determined that there is no new photo to display).

Parameters:

{number} **index**

Position in the result set of the new photo to show in the first slot (the upper-left corner). All other slots are updated accordingly. This number may be out of range of the result set and still some photos may be displayed in later slots.

**setRequest**(photoRequest)

Sets or changes the request for a widget.  
The source of photos to be displayed in the widget will be changed to this new request. The photos being shown, if any, will not be changed by this function; the user will need to manually call setPosition explicitly to show a photo from the new set, or to switch to a blank panel if 'null' was passed as the new set.

Parameters:

{panoramio.PhotoRequest|panoramio.PhotoRequestOptions|null} **photoRequest**

New request options or request object to use as a source of photos for this widget, or 'null' if the widget should be shown empty. If a panoramio.PhotoRequest object is used, the results will be shared among all widgets that use that object; if a panoramio.PhotoRequestOptions record is used, a separate request will be made for each widget.

Throws:

{panoramio.InvalidPhotoRequestError}

If 'photoRequestOptions' does not describe a valid request.