Metrici Parking Place Detector
~ Third party integration ~

Metrici Parking Place Detector (PPD) consists of one or several applications for detecting and recognising the status (occupied or free) of each monithorized parking spot. PPD engines are independent applications that analyse real time video streams from the IP cameras and send, by HTTP POSTS, the information regarding each status change of monithorized parking places, to the data display system. This system is formed of a MySQL relational data base and a collection of php scripts.

Integration with third party systems can be made by one of the following methods:

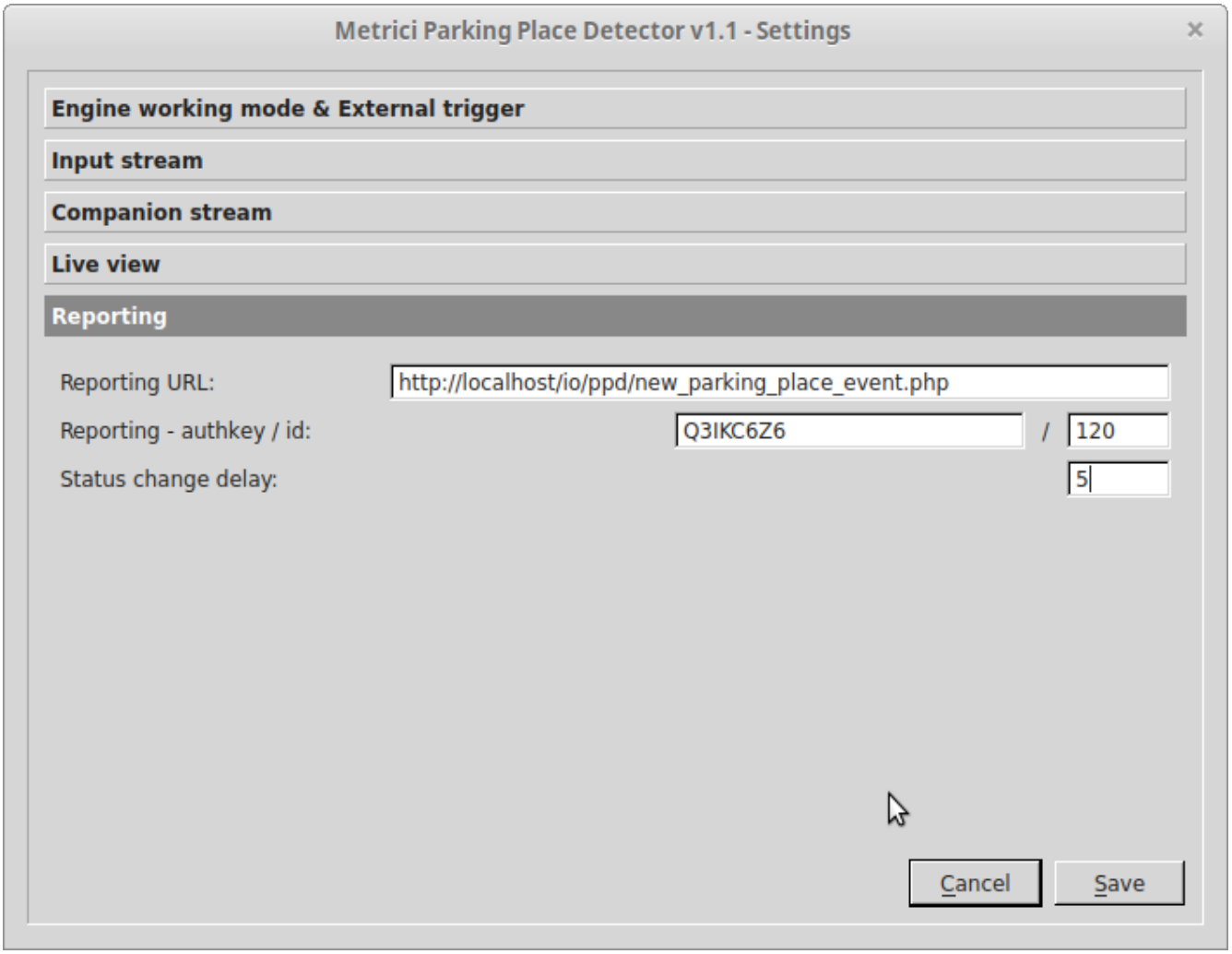
1. By PUSH directly from the PPD engines to the external system;
2. By GET from the reporting interface, using the methods from API.

1. PUSH Connection from PPD engines to the external system.

This working mode is useful when one wishes to receive parking events in real time. The drawback is that the data do not reach the web interface directly, being left to the programmer to create a retransmission mechanism for it, if necessary.

For each parking place, the PPD engine generates an event when a transition occures, from free to occupied or from occupied to free.

For each event, the PPD engine sends the data by PUSH over HTTP, in a multipart/form-data format defined in RFC2388 (http://tools.ietf.org/html/rfc2388), to an URL, which can be defined in the application interface, ‘Reporting URL’ field.



The data sent are the following:

id (int, camera id, it can be from your data base, ex: 1 = CAM1, 2 = CAM2)

space_id (int, id of the parking place, relative of this camera)

status (int, 0= free, 1= busy or occupied)

changed_at (string, yyyy-mm-dd_hh:mm:ss)

x (float, x coordinate of the parking place relative to the image width, between 0 and 1)

y (float, y coordinate of the parking place relative to the image height, between 0 and 1)

triggerkey (string, unique key which can be generated by an external trigger source)

primary_image (jpeg base64 encoded, photo of the scene)

has_companion (int, 1 if exists, 0 if there isn't defined any companion camera)

companion_image (jpeg base64 encoded, companion photo)

auth (string, md5 generated using a private key, it can be used for authentication)

auth = md5(id + space_id + status + changed_at + x + y + has_companion + reporting_check_action_authkey)

If the request is sent correctly, the external system should prompt with an approval message. If there is no answer, the PPD engine will try to resend the data subsequently. The approval message must contain the string **bb1e8f805814a0b8e465601346872377**.

Each application has a log file where all actions and statuses are submitted. All log files are located into the metrici folder and each has a name composed of the id of the application and .log extension (ex: 0.log, 1.log and so on). Below it's an example of a transaction related to an external system located at the address: http://dev2.metrici.ro/io/ppd/new_parking_place_event.php

2018-11-16 13:18:23 [Metrici Parking Place Detector v1.1] Event Free | 2018-11-16_13:18:23 | 0.46875 | 0.904105 | none for space id 5 was inserted into the local buffer
2018-11-16 13:18:23 [Metrici Parking Place Detector v1.1] Reporting event for space id 5 was sent to http://dev2.metrici.ro/io/ppd/new_parking_place_event.php
2018-11-16 13:18:23 [Metrici Parking Place Detector v1.1] Reporting reply received: bb1e8f805814a0b8e465601346872377
2018-11-16 13:33:23 [Metrici Parking Place Detector v1.1] Event Free | 2018-11-16_13:33:22 | 0.65 | 0.468632 | none for space id 13 was inserted into the local buffer
2018-11-16 13:33:23 [Metrici Parking Place Detector v1.1] Reporting event for space id 13 was sent to http://dev2.metrici.ro/io/ppd/new_parking_place_event.php
2018-11-16 13:33:23 [Metrici Parking Place Detector v1.1] Reporting reply received: bb1e8f805814a0b8e465601346872377
2018-11-16 13:34:59 [Metrici Parking Place Detector v1.1] Event Busy | 2018-11-16_13:34:58 | 0.65 | 0.468632 | none for space id 13 was inserted into the local buffer
2018-11-16 13:34:59 [Metrici Parking Place Detector v1.1] Reporting event for space id 13 was sent to http://dev2.metrici.ro/io/ppd/new_parking_place_event.php
2018-11-16 13:34:59 [Metrici Parking Place Detector v1.1] Reporting reply received: bb1e8f805814a0b8e465601346872377
2018-11-16 13:39:02 [Metrici Parking Place Detector v1.1] Event Busy | 2018-11-16_13:39:01 | 0.35125 | 0.901895 | none for space id 6 was inserted into the local buffer
2018-11-16 13:39:02 [Metrici Parking Place Detector v1.1] Reporting event for space id 6 was sent to http://dev2.metrici.ro/io/ppd/new_parking_place_event.php
2018-11-16 13:39:03 [Metrici Parking Place Detector v1.1] Reporting reply received: bb1e8f805814a0b8e465601346872377

2. Connection from the external system, by the API of the reporting interface

This working mode is useful for analyses and complex reports, subsequent to the occurrence of events, without being necessary to keep them in database located on the external system.

Using GET type requests, over HTTP, to http://IP_ADDRESS_OF_METRICI_SERVER/api, you can get data in the following formats: jSON (output=json) or XML (output = xml).

Example:

GET http://192.168.1.100/api/ppd/get_places_by_camera.php?id=14&output=json

JSON RESPONSE:

{"error":false,"response":{"body":[{"id":"1","status":"0","changed_at":"2018-11-28 17:07:21","x":"0.9675","y":"0.910737"},{"id":"2","status":"0","changed_at":"2018-11-28 18:24:52","x":"0.8775","y":"0.924"},{"id":"3","status":"0","changed_at":"2018-11-28 17:55:05","x":"0.755","y":"0.919579"},{"id":"4","status":"0","changed_at":"2018-11-28 17:16:59","x":"0.6175","y":"0.906316"},{"id":"5","status":"0","changed_at":"2018-11-28 18:29:36","x":"0.46875","y":"0.904105"},{"id":"6","status":"0","changed_at":"2018-11-28 17:59:19","x":"0.35125","y":"0.901895"},{"id":"7","status":"0","changed_at":"2018-11-28 16:52:54","x":"0.24875","y":"0.870947"},{"id":"8","status":"0","changed_at":"2018-11-28 17:34:21","x":"0.14875","y":"0.820105"},{"id":"9","status":"0","changed_at":"2018-11-28 17:35:44","x":"0.9675","y":"0.481895"},{"id":"10","status":"0","changed_at":"2018-11-28 16:17:31","x":"0.90375","y":"0.479684"},{"id":"11","status":"1","changed_at":"2018-11-28 16:14:51","x":"0.82375","y":"0.475263"},{"id":"12","status":"1","changed_at":"2018-11-28 17:24:58","x":"0.7475","y":"0.473053"},{"id":"13","status":"0","changed_at":"2018-11-28 18:29:34","x":"0.65","y":"0.468632"},{"id":"14","status":"1","changed_at":"2018-11-28 17:31:07","x":"0.565","y":"0.466421"},{"id":"15","status":"1","changed_at":"2018-11-28 07:14:15","x":"0.49125","y":"0.459789"},{"id":"16","status":"1","changed_at":"2018-11-28 06:02:16","x":"0.41875","y":"0.448737"},{"id":"17","status":"1","changed_at":"2018-11-28 04:59:57","x":"0.3575","y":"0.444316"},{"id":"18","status":"1","changed_at":"2018-11-28 15:00:36","x":"0.3","y":"0.437684"}],"header":{"duration":"0.01"}}}

Methods:

/get_locations_list – return the list of all defined locations
Entry parameters: • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_locations_list.php?output=json

/get_cameras_list – return the list of cameras defined into a location
Entry parameters: • id (integer, id of the location, obtained using /get_locations_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_cameras_list.php?id=1&output=xml

/get_groups_list – return the list of groups defined into a location
Entry parameters: • id (integer, id of the location, obtained using /get_locations_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_groups_list.php?id=1&output=xml

/get_places_by_location - return the current status (free or busy) of all parking places seen by all cameras defined into a location
Entry parameters: • id (integer, id of the location, obtained using /get_locations_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_places_by_location.php?id=1&output=json

/get_places_by_camera - return the current status (free or busy) of all parking places seen by a camera
Entry parameters: • id (integer, id of the camera, obtained using /get_cameras_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_places_by_camera.php?id=1&output=json

/get_places_by_group - return the current status (free or busy) of all parking places defined into a group
Entry parameters: • id (integer, id of the group, obtained using /get_groups_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_places_by_group.php?id=1&output=json

/get_counters_by_location - return the total, free and busy number of places defined into a location
Entry parameters: • id (integer, id of the location, obtained using /get_location_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_counters_by_location.php?id=1&output=json

/get_counters_by_camera - return the total, free and busy number of places seen by a camera
Entry parameters: • id (integer, id of the camera, obtained using /get_cameras_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_counters_by_camera.php?id=1&output=json

/get_counters_by_group - return the total, free and busy number of places defined into a group
Entry parameters: • id (integer, id of the group, obtained using /get_groups_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_counters_by_group.php?id=1&output=json

/get_status_of_place - return the current status of a particular parking place
Entry parameters: • id (integer, id of the parking place, obtained using /get_places_by_camera method) • camera_id (integer, id of the camera, obtained using /get_cameras_list method) • output (string, xml or json)
Example: http://192.168.1.100/api/ppd/get_status_of_place.php?id=1&camera_id=1&output=json

ANNEX 1 – Exemple of code, PUSH from the PPD engine, endpoint check action event

```
<?php

// v1.0
// this is given only as an example to see how one can parse and use POST
parameters reveived from Metrici PPD engine

$ok_response= "bb1e8f805814a0b8e465601346872377";
$msg= "";   //response message

//-------------------------------------
// retrieve parameters from POST

if (isset($_POST['id']))
{
      $id= (int)strip_tags($_POST['id']);
}
else
{
      $id= 0;
}

if (isset($_POST['space_id']))
{
      $space_id= (int)strip_tags($_POST['space_id']);
}
else
{
      $space_id= 0;
}

if (isset($_POST['status']))
{
      $status= (int)strip_tags($_POST['status']);
}
else
{
      $status= 0;
}

if (isset($_POST['changed_at']))
{
      $changed_at= strip_tags($_POST['changed_at']);
}
else
{
      $changed_at= "";
}

if (isset($_POST['x']))
{
      $x= (float)strip_tags($_POST['x']);
}
else
```

```
{
	$x= 0.0;
}

if (isset($_POST['y']))
{
	$y= (float)strip_tags($_POST['y']);
}
else
{
	$y= 0.0;
}

if (isset($_POST['has_companion']))
{
	$has_companion= (int)strip_tags($_POST['has_companion']);
}
else
{
	$has_companion= 0;
}

if (isset($_POST['triggerkey']))
{
	$triggerkey= strip_tags($_POST['triggerkey']);
}
else
{
	$triggerkey= "none";
}

if (isset($_POST['auth']))
{
	$auth= strip_tags($_POST['auth']);
}
else
{
	$auth= "";
}

if ($id!= 0 && $space_id!= 0 && $status!= -1 && $changed_at!= "" && $auth!= "")
	//not junk
{
	//... code  ...//
	//... retrieve authkey for this id, from your database
	//... code  ...//
	$authkey='XXXXXX';

	if($auth!= md5($id.$space_id.$status.$changed_at.$x.$y.$has_companion.
$authkey))
	{
		$msg= "Error: unauthenticated request!";
	}
	else  //all good, authenticated request
	{
		$changed_at= str_replace("_", " ", $changed_at);
```

```
            $primary_image= file_get_contents($_FILES["primary_image"]
["tmp_name"]);
            //$primary_image = mysql_real_escape_string($primary_image);
      //use it only when you want to insert the image into database

            //... code  ...//
            //... insert new data into your database
            //... code ...//

            $msg= $ok_response;
      }
}

echo $msg;  //send feedback to PPD engine

?>
```