# iPad Importer REST API

## Resources

- The following tables are accessible via the REST API
  - school
  - class
  - person
  - person_membership
  - token (this resource is different from the others, as will be seen below)
  - camera
  - deployment
  - deployment_picture
  - burst
  - image
  - tag

## Authentication and Authorization

- Every person has an email and password that they use to login.
- In order to log in, a POST is sent to /token with the following raw data:
  - { "email": "a@example.com", "password": "password" }
- A successful login will return the following JSON:
  - {"token":"X6VMY94sQRzYe2gFGdp2q3PLqAICSPiN"}
- This token should be used in **all subsequent** API as an auth header. For example:

  -
    ```
    POST /school HTTP/1.1
    Host: trap.euclidsoftware.com
    X-Trap-Token: X6VMY94sQRzYe2gFGdp2q3PLqAICSPiN
    Cache-Control: no-cache
    ```
- As you can see in the database schema, some persons are admins.  Only admins have write access to the following resources:

    - school
    - class
    - person
    - person_membership
    - camera
- Admins have write access to **all** other resources, whether they own them or not.

## Creating Resources

- In order to create a resource, send it a POST with an empty body. This will return the following JSON:
    - {"id”:”2”}
- This **id** will be used to identify this resource.  You can follow this POST with 1 or more PUT requests to create and update the resource.  For example, to create a deployment send a PUT to /deployment/2 (where 2 is the **id** returned by the POST) with the following HTTP content:

    -
      ```
      {
        "deployment_date": "1/1/2014 0:0",
        "camera" : 1,
        "nominal_mark_time" : "1/1/2014 0:0",
        "actual_mark_time" : "1/1/2014 0:0"
      }
      ```
    - Note that I have only included the required fields in this put. The rest of the fields will get default values.  If you don’t include a required field, you’ll get something like this:
        - {"message":"Required field deployment_date not specified"}
    - Of course, you do not need to specify required fields once you’re updating a record. You only need to send the fields you want to update. If you send all the fields, that’s okay too.
- The result of the above PUT call to create a deployment will be the following JSON:

    -
      ```
      {
        "deployment" : [
          {
            "deployment_date" : "2014-01-01 00:00:00",
      ```

```
    "longitude" : null,
    "short_name" : null,
    "nominal_mark_time" : "2014-01-01 00:00:00",
    "burst" : [],
    "actual_mark_time" : "2014-01-01 00:00:00",
    "camera" : 1,
    "camera_elevation_rad" : null,
    "deployment_picture" : null,
    "latitude" : null,
    "notes" : null,
    "camera_height_cm" : null,
    "id" : 2,
    "camera_azimuth_rad" : null
  }
 ]
}
```

## Updating Resources

- Send a PUT to update a resource. For example, to update the camera for deployment #2 to 2, send /deployment/2 a PUT with the following JSON:

  -
    ```
    {
     "camera" : 2
    }
    ```
- This will return you the entire record:

  -
    ```
    {
     "deployment" : [
      {
       "deployment_date" : "2014-01-01 00:00:00",
       "longitude" : null,
       "short_name" : null,
       "nominal_mark_time" : "2014-01-01 00:00:00",
       "burst" : [],
       "actual_mark_time" : "2014-01-01 00:00:00",
       "camera" : 2,
       "camera_elevation_rad" : null,
    ```

```
    "deployment_picture" : null,
    "latitude" : null,
    "notes" : null,
    "camera_height_cm" : null,
    "id" : 2,
    "camera_azimuth_rad" : null
  }
 ]
}
```

## Getting Resources

- Send a resource a GET request to retrieve it. For example, to get the record for the camera with id 1, send a GET request to /camera/1. This will return the following JSON:

  -
    ```
    {
     "camera" : [
      {
        "make" : "Trap",
        "model" : "Alpha",
        "id" : 1
      }
     ]
    }
    ```

- To get a list of all resources, send a GET request leaving off the I'd. For example. to retrieve all cameras, send a GET request to /camera.  This will return:

  -
    ```
    {
     "camera" : [
      {
        "make" : "Trap",
        "model" : "Alpha",
        "id" : 1
      },
      {
        "make" : "Trap",
        "model" : "Beta",
    ```

```
    "id" : 2
  },
  {
    "make" : "Trap",
    "model" : "Gamma",
    "id" : 3
  }
 ]
}
```

- In this version there is no way to limit the number of resources returned. This will be added in a future version.
- The API supports cascading GETs. If you request a school, the API will also return all classes associated with that school.  This is represented as school -> class.  The full list of cascading GETs is:
  - school -> class
  - class -> person
  - deployment -> deployment_picture
  - deployment -> burst
  - burst -> image
  - image -> tag
- For example, GETting /deployment/1 will return the associated bursts and deployment_pictures as well:

  -
    ```
    {
      "deployment" : [
        {
          "deployment_date" : "2014-01-01 00:00:00",
          "longitude" : null,
          "short_name" : null,
          "nominal_mark_time" : "2014-01-01 00:00:00",
          "burst" : [],
          "actual_mark_time" : "2014-01-01 00:00:00",
          "camera" : 1,
          "camera_elevation_rad" : null,
          "deployment_picture" : [
            {
              "file_name" : "myFile",
    ```

```
                "caption" : null,
                "deployment_id" : 1,
                "id" : 1,
                "description" : null
              }
            ],
            "camera_height_cm" : null,
            "notes" : null,
            "latitude" : null,
            "id" : 1,
            "camera_azimuth_rad" : null
          }
        ]
      }
```

## Images

- To create a deployment_image or picture, first create a record as you normally would and then add the associated file. To add the file, send a POST to /deployment_picture/n or image/n where n is the record id. The POST should contain a file named ‘file’. Currently only JPG files are supported.
- To view an image, send a GET to /deployment_picture/n or image/n

## Deleting Resources

- Send a DELETE request to a resource to delete it. For example sending a delete request to /image/1 will delete that image.
- Deletes will also cascade per the rules for GET above. So be careful. If you delete a school you’re deleting all the deployments and images for that school.