IOPES

Indoor-Outdoor Positioning for Emergency Staff

Grant Agreement No. 874391

D4.2: Wearable device - EMS data exchange protocol

Lead contractor: CTTC

Document status

Call (part) identifier	UCPM-2019-PP-AG
Topic	UCPM-2019-PP-PREP-AG Preparedness in civil protection and marine pollution
Grant Agreement Number	874391
Project Acronym	IOPES
Project Title	Indoor-Outdoor Positioning for Emergency Staff
Deliverable Number	4.2
Title of the Deliverable	Wearable device – EMS data exchange protocol
Work Package	WP4
Type	Report
Due date	30/09/2021
Issue date	21/10/2021
Version	1.0
Lead beneficiary	CTTC
Contributors	J. Navarro (CTTC), S. F. Hinriksson and G. Ö Guðbrandsson (SAReye)
Reviewers	E. Angelats, P. Espin, J. Navarro and M. E. Parés (CTTC)
Dissemination level	Public

This project has received funding from the European Commission, Directorate-General Humanitarian Aid and Civil Protection (ECHO), under the call UCPM-2019-PP-AG.

Revision history

Version	Date	Description
1.0	*21/10/2021*	First version of the document.

Disclaimer

Table of Contents

List of tables

List of figures

Acronyms

3G	3rd Generation (mobile telephony).
4G	4th Generation (mobile telephony).
5G	5th Generation (mobile telephony).
API	Application Programming Interface.
CPET	Civil Protection Emergency Team.
EMS	Emergency Management System.
HTTP	HyperText Transfer Protocol.
IOPES	Indoor-Outdoor Positioning for Emergency Staff.
JSON	JavaScript Object Notation.
LTE	Long Term Evolution (data transmission).
RD	Reference Document.
REST	REpresentational State Transfer (API).
RPAS	Remotely Piloted Aircraft System.
SW	Software.
URL	Uniform Resource Locator.

1. Executive summary

This document describes the mechanism that must be used to exchange data between an IOPES wearable device and an EMS (Emergency Management System).

The task of the IOPES wearable devices is to compute the positions of their users in real-time, no matter they are indoors or outdoors. Additionally, the said position must be delivered to the EMS, so the members of the emergency teams may be always tracked by the staff managing the emergency.

Therefore, and besides the need of a communication channel to make possible the remote exchange of information – such as 3G, 4G or 5G –, it is necessary to describe *how* such information will be transmitted between the wearable device and the EMS and vice versa. That is, a *protocol* must be stated to avoid misunderstanding between these two endpoints and to guarantee that information is correctly exchanged.

The task of this report is describing such protocol, implemented as a REST API (Application Programming Interface), protocol that any application – including the one driving the IOPES wearable device – must implement to make possible such communication with a target EMS.

2. Introduction

The target of the IOPES project, as already stated in other deliverables such as [RD5], is to provide Civil Protection & Emergency Teams (CPET) with the necessary tools to improve an already operational Emergency Management System (EMS) - developed by one of the partners of this project, more specifically SAReye).

Such improvements consists of, at least, (1) quickly producing and making available updated cartography of the area affected by a disaster (either natural or man-made), so it may be used in real-time (2) to track the positions of the members of the CPETs no matter whether they are located outdoors or indoors, (3) using a lightweight, portable positioning device carried by every CPET member, (4) guaranteeing that all data flows (location data) may be transmitted independently of any preexisting infrastructures thanks to the use of a portable LTE/4G easily deployable network infrastructure.

This document is focused on describing the way in which points (3) and (4) in the paragraph above have been implemented from a very specific standpoint: the way to organize data so it is correctly understood by both endpoints, that is, the positioning (wearable) device and the Emergency Management System.

Exchanging data between these to components requires a *communications channel*, usually implemented by mobile technologies such as 3G, 4G or 5G. Figure 1 depicts the structure and relationships between the several components making the IOPES concept; the said communication channel is highlighted there in red.

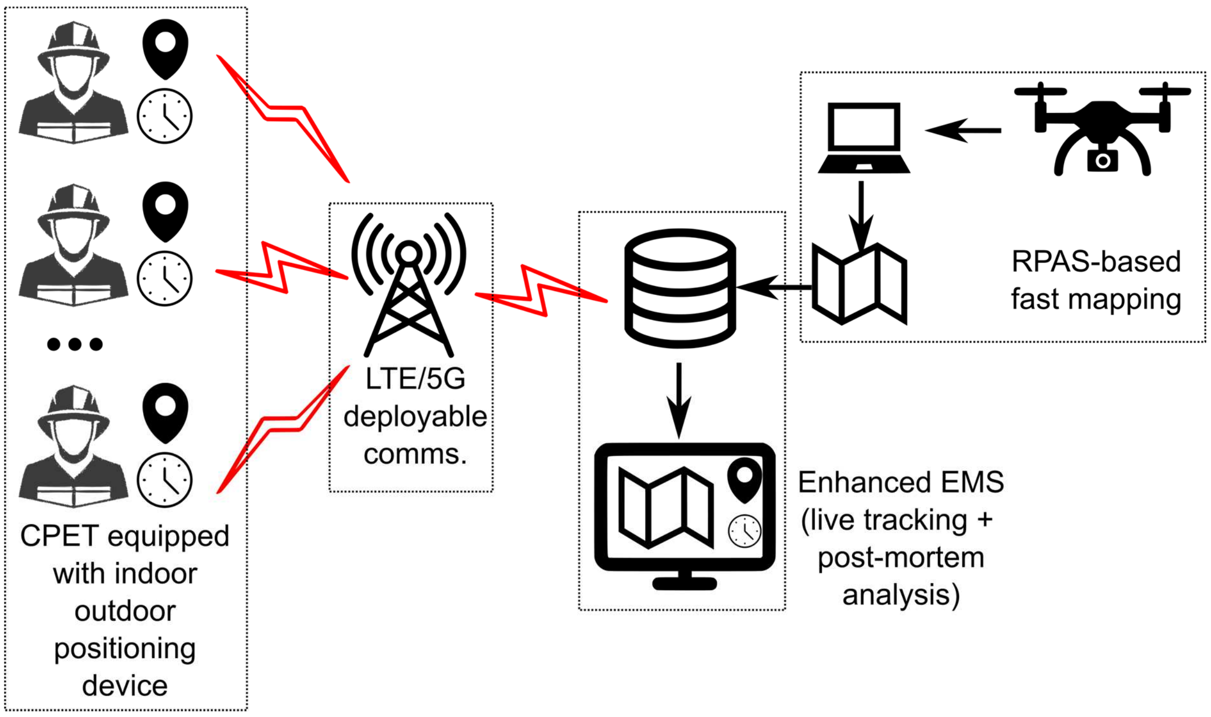

CPET equipped with indoor outdoor positioning device
LTE/5G deployable comms.
RPAS-based fast mapping
Enhanced EMS (live tracking + post-mortem analysis)

Figure 1: IOPES components. In red: data exchange between the wearable device and the EMS

It is important to note, however, that this document does not describe the necessary underlying technology required to implement these channels, such as dongles or integrated LTE/4G boards, since the said technology is considered as one more hardware component integrating the positioning device or the server where the EMS is embedded. On the contrary, the task of this report is describing how the information travelling by these channels – whatever the hardware or technology used to make possible such travel – must be organized to be understood by both endpoints.

In short, a language or *protocol* leaving no room to misinterpretation has been defined. Both clients (positioning devices, not only IOPES' but any others built by third parties) and servers (EMSs, whatever these are) must talk and understand that language to exchange positioning data correctly and timely.

The importance of this language / protocol / API lies in the fact that it opens the door to other developers willing to create their own wearable positioning devices, thus making the IOPES concept available to other projects aiming at integrating seamless indoor / outdoor positioning into other EMSs.

Section 3 describes such protocol – that is, the API – in detail but making no commitments about how to implement it. This point is capital since developers may choose between a plethora of programming languages to implement the IOPES API. Describing the API from such a neutral standpoint has a direct impact on the ease of implementation of portable positioning devices, since no restrictions are applied on how to perform such implementation.

Section 4, on the contrary, presents the specific implementation of the IOPES API that the IOPES team developed for the project and that is actually embedded in the software driving the positioning device. It has been created using the C++ programming language and it is one of the outcomes of the project. Note that this section points to an annex including the formal documentation generated by the Doxygen ([ID1]) documentation tool; therefore, the formatting and style of the annex does not adhere of the official IOPES corporate image.

3. The IOPES data exchange protocol (API)

This section presents the IOPES API from a neutral standpoint, that is, avoiding any references to the way this API should be implemented or the programming language that should be used. In this way, developers willing to interact with an IOPES-enabled EMS are free to choose the best tools to develop their client (positioning devices) applications.

However, there is a prerequisite that must be honored to guarantee that any application relying on this API will work correctly. This prerequisite states that any EMS enabling the IOPES protocol must implement a username / password / authorization token mechanism in order to identify its users and to authorize the injection of data (positions).

This means that the usual set of credentials (username plus password) normally used to grant the access to a service are not enough in this case. Instead, these credentials (1) must be used to obtain an authorization token, which must then be used (2) in all further requests to exchange data with the EMS.

Said this, the IOPES API defines only two entry points, namely “get_token” and “create_trackers”. The first entry point is used to retrieve the aforementioned authorization token; the second one, that is, “create_trackers” is the way to send positioning data to the EMS. Figure 2 depicts the typical workflow that a positioning device must follow to identify itself and then send data for as long as needed.

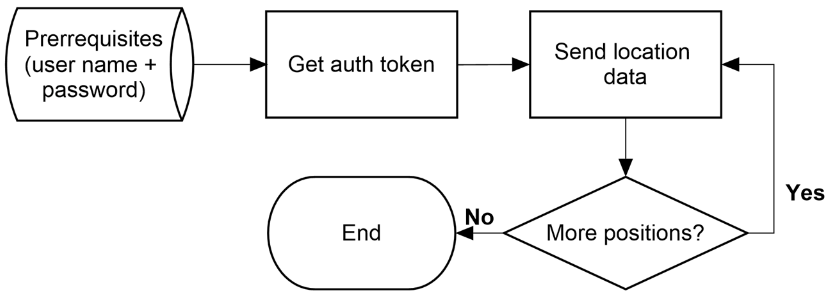


Figure 2: The IOPES API protocol

The underlying protocol on which the API relies must be https and adheres to the REST model.

1.1. The get_token entry point

The task of the get_token entry point is to obtain (return) an authorization token guaranteeing that any further data exchange request will be granted. It takes two input parameters, the username and password of the requester – who must have been previously introduced in the user’s database of the EMS.

This is the first entry point that any client application implementing the IOPES protocol must call, since it provides the necessary information (an authorization token) that will be requested in later API calls.

Table 1 describes it in detail.

Entry	get_token
Description	This endpoint provides the necessary token that should be used in all API requests for authentication purposes. The token is valid for 24 hours, after that time a new token must be requested using this same endpoint.
Method	POST
URL	https://{server_address}**/token**
Headers	None (empty)
Body (url encoded)	*key1* – string - The username of the requester. *key2* – string - The password of the requester.
Response	A JSON-formatted string containing the token sought. Its format is: {token: *the_token*}, where *the_token* is a placeholder for the actual value of the token returned by the entry point.

Table 1: The get_token entry point

Figure 3 to Figure 6 below, include some examples describing how to use this endpoint are given in a variety of programming languages.

```
curl --location --request POST 'https://server_address/token' \
--data-urlencode 'key1=some_user_name' \
--data-urlencode 'key2=the_password_for_the_user_name'
```

Figure 3: get_token - curl

```
POST /token HTTP/1.1
Host: server_address
Content-Length: 55

key1=some_user_name&key2=the_password_for_the_user_name
```

Figure 4: get_token - HTTP

```
var client = new RestClient("https://server_address/token");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
request.AddParameter("key1", "some_user_name");
request.AddParameter("key2", "the_password_for_the_user_name");
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Figure 5: get_token - C# + RestSharp

```
CURL *curl;
CURLcode res;
curl = curl_easy_init();
if(curl) {
  curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, "POST");
  curl_easy_setopt(curl, CURLOPT_URL,
"https://server_address/token");
  curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
  curl_easy_setopt(curl, CURLOPT_DEFAULT_PROTOCOL, "https");
  struct curl_slist *headers = NULL;
  curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
  const char *data =
"key1=some_user_name&key2=the_password_for_the_user_name";
  curl_easy_setopt(curl, CURLOPT_POSTFIELDS, data);
  res = curl_easy_perform(curl);
}
curl_easy_cleanup(curl);
```

Figure 6: get_token - C + library libcurl

1.2. The create_trackers entry point

Once that the authorization token has been retrieved using the endpoint get_token above, it is possible to send the position of the portable device – together with other data – to the EMS using the create_trackers endpoint.

This entry point may send an arbitrarily large number of positions at once – each of them conveniently tagged with a timestamp and some additional information – thus reducing the number of messages exchanged between client (portable device) and server (EMS). Obviously, and depending on the desired refresh frequency, it may be necessary to send one position at a time if the location of the sender must be updated as soon as possible by the receiver, not allowing for delays to reduce the amount of data transmitted over the channel.

See Table 2 for a detailed description of the create_trackers entry point.

Entry	get_token
Description	Send either a single position or an variable length array of these to the EMS. The sender must have write permissions in the server.
Method	POST
URL	https://{server_address}**/trackers**
Headers	The authorization token obtained by get_token.
Body (raw data, in JSON format)	*Trackers* – array – One or more data blocks including the positioning information. See Table 3 for details about this parameter. *user_id* – OPTIONAL uuid – Identifier of the user the trackers (positions) belongs to. If no user_id is provided, the user corresponding to the credentials provided via the get_token entry point will be used. *resource_id* – OPTIONAL uuid – Identifier of the resource the trackers belong to. Again, if no resource_id is provided, the user corresponding to the credentials provided by the get_token entry point will be used.
Response	None

Table 2: The create_trackers entry point

The trackers array consists of from 1 to *n* elements like the one described in Table 3:

Timestamp	ISO 8601 date & time in string form. The time at which this position information was obtained. See [ID2] for details on this date & time format.
latitude	A double value. The latitude in decimal degrees.
longitude	A double value. The longitude in decimal degrees.
accuracy	A double value. The radius of uncertainty for the location, measured in meters.
altitude	A double value. The altitude in meters above the WGS 84 reference ellipsoid.
altitude_accuracy	OPTIONAL – A double value. The accuracy of the altitude value, in meters.
speed	OPTIONAL – A double value. The instantaneous speed of the device in meters per second.
heading	OPTIONAL – A double value. Horizontal direction of travel of this device, measured in degrees starting at due north and continuing clockwise around the compass. Thus, north is 0 degrees, east is 90 degrees, south is 180 degrees, and so on.
device_os	OPTIONAL – A string value. The operating system of the device recording the trackers, i.e. Android, iOS, GarminOS, etc.
system_version	OPTIONAL - The version of the system or operating system of the device recording the positions.
device_information	OPTIONAL – A string value. Any additional information about the device recording the positions.

Table 3: the contents of the tracker (position) entity

The array of trackers plus the user & resource identifiers must be written as a plain text string using the JSON (see [ID3]) data model notation. The next example (see Figure 7) shows how to prepare this data, which is the contents of the body for the create_trackers entry point. Note that both the user_id and resource_id fields are optional; if the EMS interfacing with the positioning device offers no such concepts, these two items may be safely omitted. Moreover, in Figure 7, all fields have been shown, even those that are optional. Also note that, in this example, two positions have been included; should only one be sent at once, only one structure beginning with "timestamp" and ending with "device_information" would have been included.

Figure 8 to Figure 11 depict examples on how to call the create_trackers endpoint using several programming languages or tools.

```
{
	"resource_id": "the_resource_id_if_available",
	"user_id": "the_user_id_if_available",
	"trackers": [
		{
			"timestamp": "2020-04-23T11:04:29+00:00",
			"speed": 64,
			"heading": 63,
			"accuracy": 1,
			"altitude": 63,
			"device_os": "IOPES Device",
			"latitude": 64.1394958,
			"longitude": -21.907643,
			"system_version": "11.5",
			"altitude_accuracy": 10,
			"device_information": "Best tracker ever"
		},
		{
			"timestamp": "2020-04-23T11:57:29+00:00",
			"speed": 34,
			"heading": 85,
			"accuracy": 1.2,
			"altitude": 45,
			"device_os": "IOPES Device",
			"latitude": 64.05679834,
			"longitude": -22.017874,
			"system_version": "11.5",
			"altitude_accuracy": 10,
			"device_information": "Best tracker ever"
		}
	]
}
```

Figure 7: Full-blown example of the body of the create_trackers endpoint

```
curl --location --request POST 'https://server_address/trackers' \
--header 'Authorization: the_token_returned_by_get_token \
--data-raw '{
      "resource_id": "the_resource_id_if_available",
      "user_id": "the_user_id_if_available",

      "trackers": [
            {
                  "timestamp": "2020-04-23T11:04:29+00:00",
                  "speed": 64,
                  "heading": 63,
                  "accuracy": 1,
                  "altitude": 63,
                  "device_os": "IOPES Device",
                  "latitude": 64.1394958,
                  "longitude": -21.907643,
                  "system_version": "11.5",
                  "altitude_accuracy": 10,
                  "device_information": "Best tracker ever"
            },
            {
                  "timestamp": "2020-04-23T11:57:29+00:00",
                  "speed": 34,
                  "heading": 85,
                  "accuracy": 1.2,
                  "altitude": 45,
                  "device_os": "IOPES Device",
                  "latitude": 64.05679834,
                  "longitude": -22.017874,
                  "system_version": "11.5",
                  "altitude_accuracy": 10,
                  "device_information": "Best tracker ever"
            }
      ]
}'
```

Figure 8: create_trackers - curl

```
POST /trackers HTTP/1.1
Host: server_address
Authorization: the_token_returned_by_get_token
Content-Length: 708

{
      "resource_id": "the_resource_id_if_available",
      "user_id": "the_user_id_if_available",
      "trackers": [
            {
                  "timestamp": "2020-04-23T11:04:29+00:00",
                  "speed": 64,
                  "heading": 63,
                  "accuracy": 1,
                  "altitude": 63,
                  "device_os": "IOPES Device",
                  "latitude": 64.1394958,
                  "longitude": -21.907643,
                  "system_version": "11.5",
                  "altitude_accuracy": 10,
                  "device_information": "Best tracker ever"
            },
            {
                  "timestamp": "2020-04-23T11:57:29+00:00",
                  "speed": 34,
                  "heading": 85,
                  "accuracy": 1.2,
                  "altitude": 45,
                  "device_os": "IOPES Device",
                  "latitude": 64.05679834,
                  "longitude": -22.017874,
                  "system_version": "11.5",
                  "altitude_accuracy": 10,
                  "device_information": "Best tracker ever"
            }
      ]
}
```

Figure 9: create_trackers - HTTP

```
var client = new RestClient("https://server_address/trackers");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
request.AddHeader("Authorization", "the_token_returned_by_get_token");
var body = @"{" + "\n" +
@"    ""resource_id"": ""the_resource_id_if_available""," + "\n" +
@"    ""user_id"": "" he_user_id_if_available""," + "\n" +
@"    ""trackers"": [" + "\n" +
@"          {" + "\n" +
@"                ""timestamp"": ""2020-04-23T11:04:29+00:00""," + "\n"
+
@"                ""speed"": 64," + "\n" +
@"                ""heading"": 63," + "\n" +
@"                ""accuracy"": 1," + "\n" +
@"                ""altitude"": 63," + "\n" +
@"                ""device_os"": ""IOPES Device""," + "\n" +
@"                ""latitude"": 64.1394958," + "\n" +
@"                ""longitude"": -21.907643," + "\n" +
@"                ""system_version"": ""11.5""," + "\n" +
@"                ""altitude_accuracy"": 10," + "\n" +
@"                ""device_information"": ""Best tracker ever""" + "\n"
+
@"          }," + "\n" +
@"          {" + "\n" +
@"                ""timestamp"": ""2020-04-23T11:57:29+00:00""," + "\n"
+
@"                ""speed"": 34," + "\n" +
@"                ""heading"": 85," + "\n" +
@"                ""accuracy"": 1.2," + "\n" +
@"                ""altitude"": 45," + "\n" +
@"                ""device_os"": ""IOPES Device""," + "\n" +
@"                ""latitude"": 64.05679834," + "\n" +
@"                ""longitude"": -22.017874," + "\n" +
@"                ""system_version"": ""11.5""," + "\n" +
@"                ""altitude_accuracy"": 10," + "\n" +
@"                ""device_information"": ""Best tracker ever""" + "\n"
+
@"          }" + "\n" +
@"    ]" + "\n" +
@"}";
request.AddParameter("text/plain", body,  ParameterType.RequestBody);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Figure 10: create_trackers - C# + RestSharp

```
CURL *curl;
CURLcode res;
curl = curl_easy_init();
if(curl) {
  curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, "POST");
  curl_easy_setopt(curl, CURLOPT_URL, "https://server_address
/trackers");
  curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
  curl_easy_setopt(curl, CURLOPT_DEFAULT_PROTOCOL, "https");
  struct curl_slist *headers = NULL;
  headers = curl_slist_append(headers, "Authorization:
the_token_returned_by_get_token");
  curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
  const char *data = "{\n     \"resource_id\": \"
the_resource_id_if_available \",\n  \"user_id\": \"
the_user_id_if_available \",\n      \"trackers\": [\n       {\n
      \"timestamp\": \"2020-04-23T11:04:29+00:00\",\n
      \"speed\": 64,\n             \"heading\": 63,\n
      \"accuracy\": 1,\n                \"altitude\": 63,\n
      \"device_os\": \"IOPES Device\",\n            \"latitude\":
64.1394958,\n               \"longitude\": -21.907643,\n
      \"system_version\": \"11.5\",\n
      \"altitude_accuracy\": 10,\n             \"device_information\":
\"bla\"\n         },\n        {\n              \"timestamp\": \"2020-04-
23T11:57:29+00:00\",\n              \"speed\": 64,\n
      \"heading\": 63,\n                \"accuracy\": 1,\n
      \"altitude\": 63,\n               \"device_os\": \"IOPES
Device\",\n             \"latitude\": 64.0494958,\n
      \"longitude\": -21.907643,\n             \"system_version\":
\"11.5\",\n             \"altitude_accuracy\": 10,\n
      \"device_information\": \"bla\"\n        }\n    ]\n}";
  curl_easy_setopt(curl, CURLOPT_POSTFIELDS, data);
  res = curl_easy_perform(curl);
}
curl_easy_cleanup(curl);
```

Figure 11: create_trackers - C + library libcurl

4. The C++ implementation of the IOPES API

The team involved in the IOPES project has developed a working C++ implementation of the IOPES API described in section 3. It is, in fact, part of the software driving the IOPES portable positioning device, so it has been tested in real life environments and shown that it is fully operational.

It relies in two external libraries, namely libcurl ([ID4]) and SimpleJSON ([ID5]) to implement the file transfer over the internet and the coding and decoding of JSON strings respectively.

Note that there are two alternative classes to work (1) with files – just for software testing purposes – and (2) over the internet, actually exchanging data with an IOPES capable EMS implementing its side of the entry points defined by the API.

Offering a class interacting with files instead of the actual EMS IOPES-capable servers is an extra feature allowing developers to test the software in their portable positioning devices without having to rely on neither in existing communication channels nor in an EMS willing to accept fake, test data.

The IOPES_API library has been thoroughly documented using the Doxygen ([ID1]) tool. Such documentation is appended as an annex in section 7; note, however, that since it has been generated with the automated tool Doxygen, the format of the resulting document does not adhere to the style guides set by the IOPES consortium.

5. Reference documents

[RD1] IOPES – Grant Agreement (GA) – GA 874391.
[RD2] IOPES – Consortium Agreement (CA) – Version 1.0.
[RD3] Union Civil Protection Mechanism. Prevention and Preparedness Projects in Civil Protection and Marine Pollution. Call for proposals document UCPM-2019-PP-AG - Version 1.0.
[RD4] IOPES – Deliverable D3.1, "User requirements".
[RD5] IOPES – Deliverable D4.1. "System architecture definition".

6. Informative documents

[ID1] Doxygen. https://www.doxygen.nl/index.html (20th October 2021).
[ID2] ISO 8601:2004. https://www.iso.org/standard/40874.html . (20th October 2021).
[ID3] ECMA-404 The JSON Data Interchange Standard. https://www.json.org/json-en.html (20th October 2021).
[ID4] libcurl – the multiprotocol file transfer library. https://curl.se/libcurl/ . (20th October 2021).
[ID5] SimpleJSON library. https://github.com/MJPA/SimpleJSON . (20th October 2021).

7. Annex: Doxygen documentation

The next and following pages (up to the end of this document) contain the Doxygen documentation for the C++ implementation of the IOPES API (see section 4).

IOPES_API

version 1.0

Generated by Doxygen 1.9.1

Chapter 1

The IOPES project team's C++ implementation of the IOPES API

This is the IOPES project team's C++ implementation of the IOPES API as defined in the document "D4.2 - Wearable device - EMS data exchange protocol". More information about the IOPES project may be found here: `https://iopes-project.eu/`.

The code described here has been organized in a very simple set of classes:

- IOPES_API
  - IOPES_API_file
  - IOPES_API_curl

The first class in the list above (IOPES_API) is an abstract one, so it must never be directly instantiated. It is provided just to define the set of methods and common attributes that any descendant class must implement / have.

On the contrary, IOPES_API_file is a fully usable class targeted at debugging code. This pseudo-implementation of the IOPES API simulates the protocol there defined, providing the whole set of method (API entry points). However, these methods are fake, since either do nothing (get_token()) or log data to a disk file instead of sending the information to an IOPES-enabled EMS server. The idea behind this class is providing a mechanism to test the software using the IOPES API, providing "real" IOPES API calls, freeing the developers of the need of having a full-fledged server accepting their requests. The information "sent" by create_trackers() is logged in a disk file, being possible to check whether the appropriate information is "transmitted".

Finally, class IOPES_API_curl is a full-fledged implementation of the protocol relying on the curl (aka libcurl) library. In this case, an operative, IOPES-enabled EMS is required to run the software using this class.

This C++ implementation of the IOPES API relies in two libraries:

- `SimpleJSON`. It is used to code / decode JSON strings that are exchanged between the client and the server. The source code for this library is packed along with this implementation of the IOPES API for simplicity reasons. Should any newer version of this library be used, then it would be required to download the new fonts from the URL above.
- `libcurl`. Used to actually exchange data between the client and the EMS. This library is not included in the package and must be installed before using this one.

A full example using the IOPES_API_curl class may be found in A full working example.

Chapter 2

A full working example

2.1 About the example

The following example shows how to exchange data between a portable device (or any other client) and a IOPES-enabled EMS.

A few comments on the example:

- It uses the IOPES_API_curl class to communicate with an EMS, actually sending data. If no EMS is available, it is possible to use files following the next steps:
  - Change sentence '#include "IOPES_API_curl.hpp"' to '#include "IOPES_API_file.hpp"'.
  - Change the declaration "IOPES_API_curl api;" to "IOPES_API_file api;".
  - Change the sentence 'URL = "https://cd18c941-a793-4a4c-bdb4-692b0091f7d9.mock.pstmn.io";' to something similar to 'URL = "path_to_some_file";' (where "path_to_some_file" actually points to a disk file to write data).
- The data to be sent to the EMS is a fake. The first part of the example program builds synthetic positions instead of capturing these from any positioning device.
- Just a set of two positions is sent (at once). A real program would iterate fetching positions and sending these to some EMS.
- The URL used in the example points to some server that was willing to accept connections at some time. Change the URL to that of your server. Note that it must be IOPES-enabled (that is, willing to accept get_↩ token() and create_trackers() requests)-

2.2 The full example

```
// The next header file includes the definitions required to use
// the curl-based implementation of the IOPES API.
#include "IOPES_API_curl.hpp"
// This one is required for "cout".
#include <iostream>
using namespace std;
int
main
(void)
{
  {
    string                    password;
    string                    resource_id;
    int                       status;
    string                    token;
```

```
vector<IOPES_API_tracker> trackers;
string                    URL;
string                    user_id;
string                    username;
// ------------------------------------------------------------------------
//
// Build some FAKE data to show how to use the interface.
IOPES_API_tracker tracker1;
IOPES_API_tracker tracker2;
// Both the user and resource ids are OPTIONAL. If you don't have any,
// simply set these to the empty string ("").
user_id     = "This is the user id";
resource_id = "This is the resource id";
//
// The "tracker<x>" structures hold the several positions that we
// have retrieved. In this example, we set two of these at once.
// Note that the first tracker (position) includes the whole set
// of data that may be transferred to the server, while the second
// one contains only the minimum fields (the mandatory ones).
//
// When an optional string is not available, just set it to the
// empty string (""). When an optional field is not available,
// set it to any value, but set the corresponding got_<fieldname>
// field to false.
//
// On the contrary, when an optional string is available, set it
// to the value it must hold, and for optional numerical fields,
// set these to the value they must contain and the corresponding
// got_<fieldname> to true.
//
tracker1.date_time      = "2020-11-02T06:18:24+00:00";
tracker1.longitude      = 1.2345;
tracker1.latitude       = 6.7890;
tracker1.xy_accuracy    = 1.2;
tracker1.altitude       = 4.56789;
tracker1.got_z_accuracy = true;
tracker1.z_accuracy     = 1.2345678;
tracker1.speed          = 901.234567;
tracker1.got_speed      = true;
tracker1.heading        = 89012.3456;
tracker1.got_heading    = true;
tracker1.device_os      = "IOPES wearable";
tracker1.system_version = "1.0";
tracker1.device_info    = "Serial number 1234567";
tracker2.date_time      = "2020-11-02T06:18:24+01:00";
tracker2.longitude      = -180.123456;
tracker2.latitude       = 89.1234567;
tracker2.xy_accuracy    = 8.9;
tracker2.altitude       = 123.456789;
tracker2.z_accuracy     = 1.2345678;  // Meaningless, see got_z_accuracy
                                      // below.
tracker2.got_z_accuracy = false;
tracker2.speed          = 333.333333; // Meaningless, see got_speed below.
tracker2.got_speed      = false;
tracker2.heading        = 444.444444; // Meaningless, see got_heading below.
tracker2.got_heading    = false;
tracker2.device_os      = "";         // No device_os available ("").
tracker2.system_version = "";         // No system_version available ("").
tracker2.device_info    = "";         // No device_info available ("").
// Add the trackers to the trackers array.
trackers.push_back(tracker1);
trackers.push_back(tracker2);
//
// ------------------------------------------------------------------------
// This is our object to deal with the IOPES API.
IOPES_API_curl api;
//
// Set the URL of the server. Replace the URL below by the one you have
// to use to contact YOUR server. DON'T append the entry points (such as
// "/token" or "/trackers" to the URL. The IOPES_API_curl object will
// take care of this by itself.
//
URL = "https://cd18c941-a793-4a4c-bdb4-692b0091f7d9.mock.pstmn.io";
api.set_channel(URL);
//
// We'll request now an authorization token using our username & password.
// Replace the values below by those given by your EMS provider.
//
username = "this_is_the_username";
password = "this_is_the_password";
status = api.get_token(username, password, token);
// Check whether we succeeded or not.
if (status != 0)
{
  // Bad luck, we've got problems.
  cout « "get_token returned an error code: " « status « endl;
  return -1;
```

```
    }
    else
    {
      // Oh, yes! We’ve got an authorization token!
      cout « "The token retrieved is ’" « token « "’" « endl;
    }
    //
    // Now, create trackers (send positions to the server). Note that:
    // - We’re using the token just obtained,
    // - We always pass the user_id and resource_id, even when we don’t
    //   have these (in such cases, set them to the empty string, "").
    // - We pass a whole array of trackers (positions). Such array my
    //   contain just one tracker object, if we prefer to do it this
    //   way.
    //
    // This step should be repeated whilst there are positions (trackers)
    // to send to the server.
    //
    status = api.create_trackers(token, user_id, resource_id, trackers);
    if (status != 0)
    {
      cout « "create_trackers returned an error code: " « status « endl;
      return -1;
    }
    else
    {
      cout « "create_trackers succeeded." « endl;
    }
    // That’s all.
    return 0;
  }
}
```

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

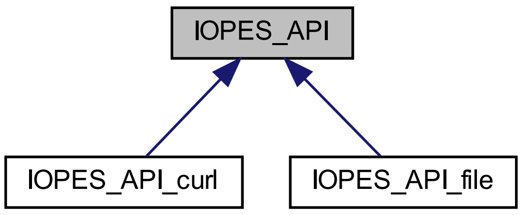
Class Documentation

6.1 IOPES_API Class Reference

Abstract class definining the entry points implementing the IOPES API, no matter the device used to send data.

```
#include <IOPES_API.hpp>
```

Inheritance diagram for IOPES_API:



Public Member Functions

- virtual int create_trackers (const string &token, const string &user_id, const string &resource_id, const vector< IOPES_API_tracker > &trackers)=0

  Create trackers - notify location data.
- virtual int get_token (const string &username, const string &password, string &token)

  Get the authorization token.
- IOPES_API (void)

  Default constructor.
- virtual int set_channel (const string &channel_id)=0

  Set the address / name / path of the resource used to send / store data.
- virtual ~IOPES_API (void)

  Destructor.

Protected Attributes

- string channel_id_

  The identification of the channel / resource used to send or store data. Heir classes may interpret this resource as a file name, URL or whatever kind of resource they use.

6.1.1 Detailed Description

Abstract class definining the entry points implementing the IOPES API, no matter the device used to send data.

This class is the base to implement the IOPES API (defined in IOPES' deliverable "D4.2: Wearable device - EMS data exchange protocol").

This one is just an abstract class setting the framework that all derived classes will have to implement. Therefore, it should never be instantiated. Instead, the heir classes are the ones to use.

Each descendant class must implement a different way (i.e., using different file transfer technologies, such as the curl library).

6.1.2 Member Function Documentation

6.1.2.1 create_trackers()

```
virtual int IOPES_API::create_trackers (
            const string & token,
            const string & user_id,
            const string & resource_id,
            const vector< IOPES_API_tracker > & trackers )  [pure virtual]
```

Create trackers - notify location data.

Parameters

token	The authorization token retrieved by get_token().
user_id	The user the trackers belon to. OPTIONAL: set it to the empty string ("") when no user id is available.
resource←_id	Identifier of the resource the trackers belong to. OPTIONAL: set it to the empty string ("") when no resource id is available.
trackers	The set of trackers (positions) to report.
response	The response received once the trackers have been reported.

Returns

An error code. Heir classes may extend the list of return codes, but at least, the "successful completion" one (value: 0) must be present:

- 0: Successful completion.

Implemented in IOPES_API_file, and IOPES_API_curl.

6.1.2.2 get_token()

```
int IOPES_API::get_token (
             const string & username,
             const string & password,
             string & token )  [virtual]
```

Get the authorization token.

Parameters

username	The user name used to retrieve the authorization token.
password	The password of the user used to retrieve the authorization token.
token	Output. The requested authorization token.

Returns

An error code. Heir classes may define extra return codes, but the implementation for this parent class just return one:

- 0: Successful completion.

Reimplemented in IOPES_API_curl.

6.1.2.3 set_channel()

```
virtual int IOPES_API::set_channel (
             const string & channel_id )  [pure virtual]
```

Set the address / name / path of the resource used to send / store data.

This method must be used by all descendant classes to define the "destination" where the information will be retrieved from (using get_token()) or sent to (by means of create_trackers()).

Depending on the descendant class used, this "channel_id" may take several forms, such as a file path or a URL. Check the documentation for the particular descendant classes to learn what kind of channel identifier is expected.

Parameters

channel↩_id	Path, URL or whatever kind of resource id to identify the channel used to send data.

Returns

An error code. At least, the "successful completion" code with a return value 0 must be implemented. Heir classes may extend this list of codes.

- 0: Successful completion.

Implemented in IOPES_API_file, and IOPES_API_curl.

The documentation for this class was generated from the following files:

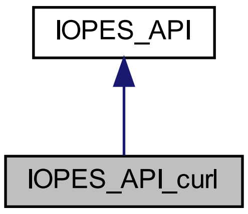
- IOPES_API.hpp
- IOPES_API.cpp

6.2 IOPES_API_curl Class Reference

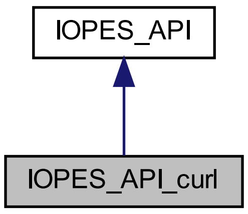
Implementation of the IOPES API using the curl library.

```
#include <IOPES_API_curl.hpp>
```

Inheritance diagram for IOPES_API_curl:



Collaboration diagram for IOPES_API_curl:



Public Member Functions

- virtual int create_trackers (const string &token, const string &user_id, const string &resource_id, const vector< IOPES_API_tracker > &trackers) override

  Create trackers - notify location data.
- virtual int get_token (const string &username, const string &password, string &token) override

  Get the authorization token.
- IOPES_API_curl (void)

  Default constructor.
- virtual int set_channel (const string &channel_id) override

  Set the path to the file where data will be written. Open the file.
- virtual ~IOPES_API_curl (void)

  Destructor.

Protected Member Functions

- string tracker_to_json_string (const IOPES_API_tracker &tracker)

  Format the contents of a single tracker structure as a JSON string adhering to the IOPES' API specs.
- string trackers_to_json (const string &user_id, const string &resource_id, const vector< IOPES_API_tracker > &trackers)

  Format a whole set of tracker structures plus the optional user and resource identifiers as a JSON string adhering to the IOPES' specs.

Protected Attributes

- string channel_id_

  The identification of the channel / resource used to send or store data. Heir classes may interpret this resource as a file name, URL or whatever kind of resource they use.
- string URL_token_

  The URL for the "token" entry point.
- string URL_trackers_

  The URL for the "create_trakers" entry point.

6.2.1 Detailed Description

Implementation of the IOPES API using the curl library.

This class implements the IOPES API using the curl (aka "libcurl"). It also realies on the SimpleJSON library to code / decode JSON strings. Developers need, therefore, to guarantee that these two libraries are available when compiling this code.

See IOPES' deliverable "D4.2: Wearable device - EMS data exchange protocol" for more information about where to obtain both libcurl and SimpleJSON.

As a fully functional class, an IOPES-enabled EMS must be available to run code relying on this class. For debugging purposes, requiring no such server, see IOPES_API_file.

6.2.2 Member Function Documentation

6.2.2.1 create_trackers()

```
int IOPES_API_curl::create_trackers (
            const string & token,
            const string & user_id,
            const string & resource_id,
            const vector< IOPES_API_tracker > & trackers ) [override], [virtual]
```

Create trackers - notify location data.

Parameters

token	The authorization token retrieved by get_token().
user_id	The user the trackers belong to. OPTIONAL: set it to the empty string ("") when no user id is available.
resource←_id	Identifier of the resource the trackers belong to. OPTIONAL: set it to the empty string ("") when no resource id is available.
trackers	The set of trackers (positions) to report.

Returns

An error code.

- 0: Successful completion.
- 1: Unable to set up the curl library.
- 2: Error sending tracker data to the server.

Implements IOPES_API.

6.2.2.2 get_token()

```
int IOPES_API_curl::get_token (
            const string & username,
            const string & password,
            string & token ) [override], [virtual]
```

Get the authorization token.

Parameters

username	The user name used to retrieve the authorization token.
password	The password of the user used to retrieve the authorization token.
token	Output. The requested authorization token.

Returns

An error code.

- 0: Successful completion.

- 1: Unable to set up the curl library.
- 2: Unable to retrieve the token (connection problems).
- 3: Malformed response received from the server. Unable, therefore, to retrieve the token.

Reimplemented from IOPES_API.

6.2.2.3 set_channel()

```
int IOPES_API_curl::set_channel (
            const string & channel_id )  [override], [virtual]
```

Set the path to the file where data will be written. Open the file.

This class requires URLs to identify the target server. See parameter channel_id.

Parameters

channel←_id	URL (including the preceding protocol such as "https://") of the IOPES-enabled EMS to connect to.

Returns

An error code that will always be zero, since setting an URL will never fail. The returned error code is provided for compability reasons only.

- 0: Successful completion.

Implements IOPES_API.

6.2.2.4 tracker_to_json_string()

```
string IOPES_API_curl::tracker_to_json_string (
            const IOPES_API_tracker & tracker )  [protected]
```

Format the contents of a single tracker structure as a JSON string adhering to the IOPES' API specs.

Parameters

user_id	The user the trackers belong to. OPTIONAL: set it to the empty string ("") when no user id is available.
resource←_id	Identifier of the resource the trackers belong to. OPTIONAL: set it to the empty string ("") when no resource id is available.
tracker	The tracker whose values have to be formatted as a JSON string.

6.2.2.5 trackers_to_json()

```
string IOPES_API_curl::trackers_to_json (
            const string & user_id,
            const string & resource_id,
            const vector< IOPES_API_tracker > & trackers )  [protected]
```

Format a whole set of tracker structures plus the optional user and resource identifiers as a JSON string adhering to the IOPES' specs.

Parameters

user_id	The user identifier. OPTIONAL. Set it to the empty string ("") when no user identifier is available.
resource←_id	The resource identifier. OPTIONAL. Set it to the empty string ("") when no resource identifier is available.
The	set of trackers that must be formatted. Note that the array containing the trackers may have just one element, if desired.

Returns

A string, formatted as JSON, containing the whole dataset formatted according to IOPES' API needs.

The documentation for this class was generated from the following files:

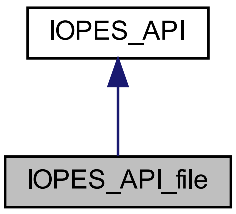
- IOPES_API_curl.hpp
- IOPES_API_curl.cpp

6.3 IOPES_API_file Class Reference

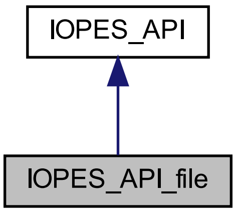
Implementation of the IOPES API using files as the backend. This class will must be used for testing purposes ONLY.

```
#include <IOPES_API_file.hpp>
```

Inheritance diagram for IOPES_API_file:



Collaboration diagram for IOPES_API_file:



Public Member Functions

- virtual int create_trackers (const string &token, const string &user_id, const string &resource_id, const vector< IOPES_API_tracker > &trackers) override

    Create trackers - notify location data.
- virtual int get_token (const string &username, const string &password, string &token)

    Get the authorization token.
- IOPES_API_file (void)

    Default constructor.
- virtual int set_channel (const string &channel_id) override

    Set the path to the file where data will be written. Open the file.
- virtual ~IOPES_API_file (void)

    Destructor.

Protected Attributes

- string channel_id_

    The identification of the channel / resource used to send or store data. Heir classes may interpret this resource as a file name, URL or whatever kind of resource they use.
- ofstream ofile_

    The file to write data to.
- bool ready_

    Flag stating whether the output file is open.

6.3.1 Detailed Description

Implementation of the IOPES API using files as the backend. This class will must be used for testing purposes ONLY.

This class emulates the IOPES_API using files to incarnate the remote EMS.

The only goal of this class is to provide developers with an IOPES API - compatible class requiring no remote servers, so when developing software for a portable positioning device, it will be possible to simulate that the transmission of data is taking place.

Instead of sending the said data to some EMS server, it is written to files, so it is possible to check if the information "sent" to the fake server (the output file) has been correctly "received".

Note that since no authorization token may be ever provided by a file system, the get_token() method is not overriden here. Therefore, users of this class will use this parent class' get_token() method implicitly.

6.3.2 Member Function Documentation

6.3.2.1 create_trackers()

```
int IOPES_API_file::create_trackers (
            const string & token,
            const string & user_id,
            const string & resource_id,
            const vector< IOPES_API_tracker > & trackers )  [override], [virtual]
```

Create trackers - notify location data.

This method will never complain about the validity of the token parameter, since interacting with a file system implies that it will never be possible to obtain an "appropriate" one. Therefore, pass whatever value for this parameter (even the one returned by this class parent's get_token() method.

Parameters

token	The authorization token retrieved by get_token(). Use this parent class' get_token() to retrieve it.
user_id	The user the trackers belon to. OPTIONAL: set it to the empty string ("") when no user id is available.
resource↩ _id	Identifier of the resource the trackers belong to. OPTIONAL: set it to the empty string ("") when no resource id is available.
trackers	The set of trackers (positions) to report.

Returns

An error code.

- 0: Successful completion.
- 1: Error writing the trackers to the output file: file not open.
- 2: Error writing to the output file.

Implements IOPES_API.

6.3.2.2 get_token()

```
int IOPES_API::get_token (
            const string & username,
            const string & password,
            string & token )  [virtual], [inherited]
```

Get the authorization token.

Parameters

username	The user name used to retrieve the authorization token.
password	The password of the user used to retrieve the authorization token.
token	Output. The requested authorization token.

Returns

An error code. Heir classes may define extra return codes, but the implementation for this parent class just return one:

- 0: Successful completion.

Reimplemented in IOPES_API_curl.

6.3.2.3 set_channel()

```
int IOPES_API_file::set_channel (
            const string & channel_id )  [override], [virtual]
```

Set the path to the file where data will be written. Open the file.

This class uses paths to files on disk to play the role of the "channel_id".

Parameters

channel↔_id	Path to the file where data will be written.

Returns

An error code.

- 0: Successful completion.
- 1: Error opening the output file.

Implements IOPES_API.

The documentation for this class was generated from the following files:

- IOPES_API_file.hpp
- IOPES_API_file.cpp

6.4 IOPES_API_response Struct Reference

Structure used to retrieve responses after calling an entry point.

```
#include <IOPES_API_structures.hpp>
```

Public Attributes

- char ∗ response_data
- size_t size

6.4.1 Detailed Description

Structure used to retrieve responses after calling an entry point.

6.4.2 Member Data Documentation

6.4.2.1 response_data

```
char* IOPES_API_response::response_data
```

The response's text.

6.4.2.2 size

```
size_t IOPES_API_response::size
```

The number of bytes (size of) in the response.

The documentation for this struct was generated from the following file:

- IOPES_API_structures.hpp

6.5 IOPES_API_tracker Struct Reference

Structure defining a tracker (position)

```
#include <IOPES_API_structures.hpp>
```

Public Attributes

- double altitude
- string date_time
- string device_info
- string device_os
- bool got_heading
- bool got_speed
- bool got_z_accuracy
- double heading
- double latitude
- double longitude
- double speed
- string system_version
- double xy_accuracy
- double z_accuracy

6.5.1 Detailed Description

Structure defining a tracker (position)

6.5.2 Member Data Documentation

6.5.2.1 altitude

```
double IOPES_API_tracker::altitude
```

The altitude in meters above the WGS 84 reference ellipsoid.

6.5.2.2 date_time

```
string IOPES_API_tracker::date_time
```

The time at which this position information was obtained, in ISO 8601 format.

6.5.2.3 device_info

```
string IOPES_API_tracker::device_info
```

OPTIONAL. Any additional information of the device recording data. Set it to the empty string ("") when no additional information about the device is available.

6.5.2.4 device_os

```
string IOPES_API_tracker::device_os
```

OPTIONAL. The operating system of the device the trackers, i.e. Android, iOS, GarminOS, etc. Set it to the empty string ("") when no device OS information is available.

6.5.2.5 got_heading

```
bool IOPES_API_tracker::got_heading
```

Flag stating whether the optional field heading holds a meaningful value.

6.5.2.6 got_speed

```
bool IOPES_API_tracker::got_speed
```

Flag stating whether the optional field holds a meaningful value.

6.5.2.7 got_z_accuracy

```
bool IOPES_API_tracker::got_z_accuracy
```

Flag stating whether the optional field z_accuracy holds a valid value.

6.5.2.8 heading

```
double IOPES_API_tracker::heading
```

OPTIONAL. Horizontal direction of travel of this device, measured in degrees starting at due north and continuing clockwise around the compass. Thus, north is 0 degrees, east is 90 degrees, south is 180 degrees, and so on. Valid only when got_heading is true.

6.5.2.9 latitude

```
double IOPES_API_tracker::latitude
```

The longitude in decimal degrees.

6.5.2.10 longitude

```
double IOPES_API_tracker::longitude
```

The latitude in decimal degrees.

6.5.2.11 speed

```
double IOPES_API_tracker::speed
```

OPTIONAL. The instantaneous speed of the device in meters per second. Valid only when got_speed is true.

6.5.2.12 system_version

```
string IOPES_API_tracker::system_version
```

OPTIONAL. The version of the system or operating system of the device recording the trackers. Set it to the empty string ("") when no information about the system's version is available.

6.5.2.13 xy_accuracy

```
double IOPES_API_tracker::xy_accuracy
```

The radius of uncertainty for the location, measured in meters.

6.5.2.14 z_accuracy

```
double IOPES_API_tracker::z_accuracy
```

OPTIONAL. The accuracy of the altitude value, in meters. Valid only when got_z_accuracy is true.

The documentation for this struct was generated from the following file:

- IOPES_API_structures.hpp

Chapter 7

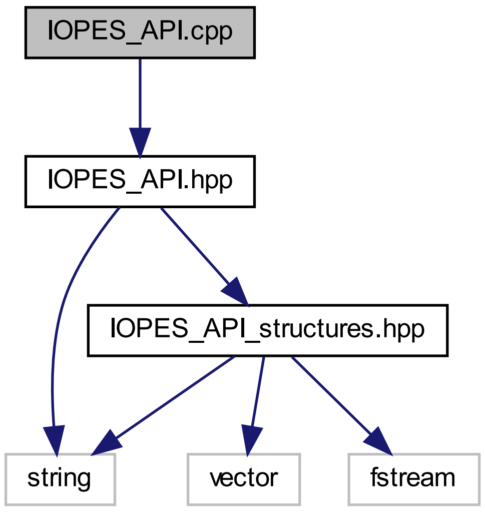
File Documentation

7.1 IOPES_API.cpp File Reference

Implementation file for IOPES_API.hpp.

```
#include "IOPES_API.hpp"
```

Include dependency graph for IOPES_API.cpp:



7.1.1 Detailed Description

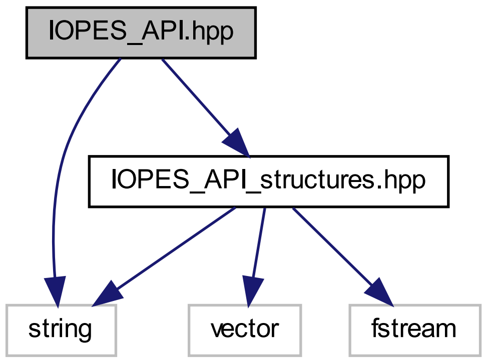
Implementation file for IOPES_API.hpp.

7.2 IOPES_API.hpp File Reference

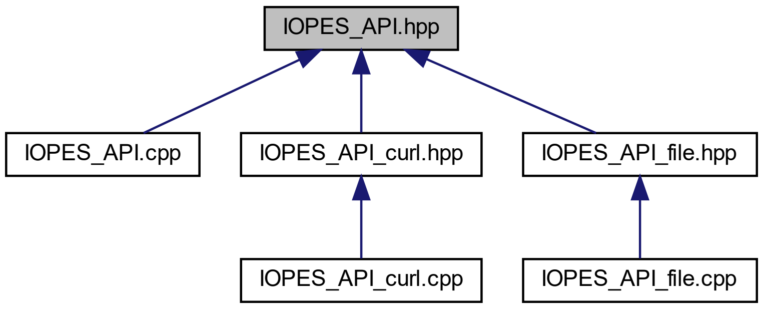
Abstract class definining the entry points implementing the IOPES API, no matter the device used to send data.

```
#include "IOPES_API_structures.hpp"
#include <string>
```

Include dependency graph for IOPES_API.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class IOPES_API

  Abstract class definining the entry points implementing the IOPES API, no matter the device used to send data.

7.2.1 Detailed Description

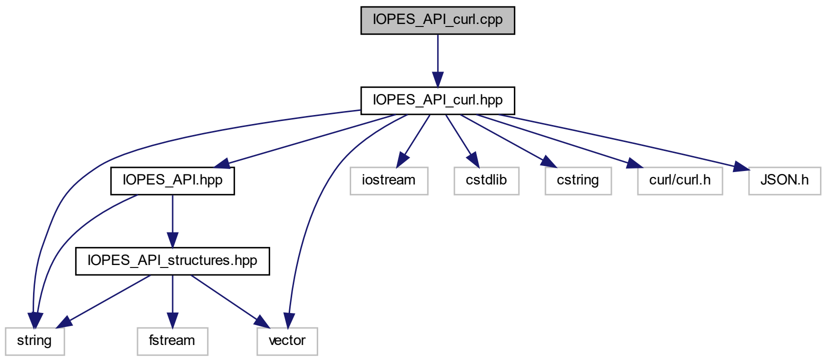
Abstract class definining the entry points implementing the IOPES API, no matter the device used to send data.

7.3 IOPES_API_curl.cpp File Reference

Implementation file for IOPES_API_curl.cpp.

```
#include "IOPES_API_curl.hpp"
```

Include dependency graph for IOPES_API_curl.cpp:



7.3.1 Detailed Description

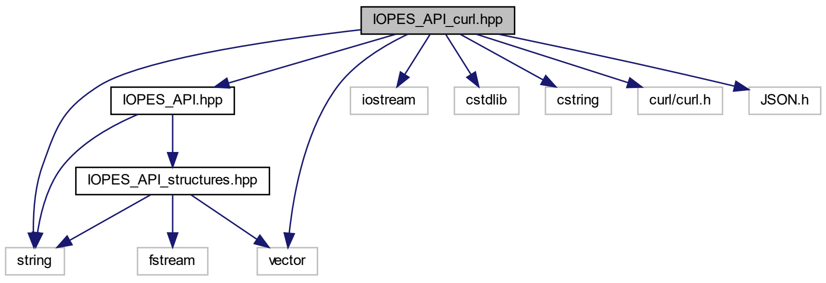
Implementation file for IOPES_API_curl.cpp.

7.4 IOPES_API_curl.hpp File Reference

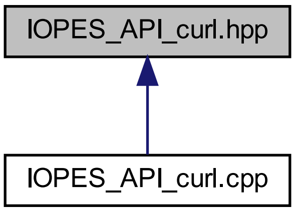
Implementation of the IOPES API using the curl library.

```
#include "IOPES_API.hpp"
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <string>
#include <vector>
#include <curl/curl.h>
#include "JSON.h"
```

Include dependency graph for IOPES_API_curl.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class IOPES_API_curl

  Implementation of the IOPES API using the curl library.

7.4.1 Detailed Description

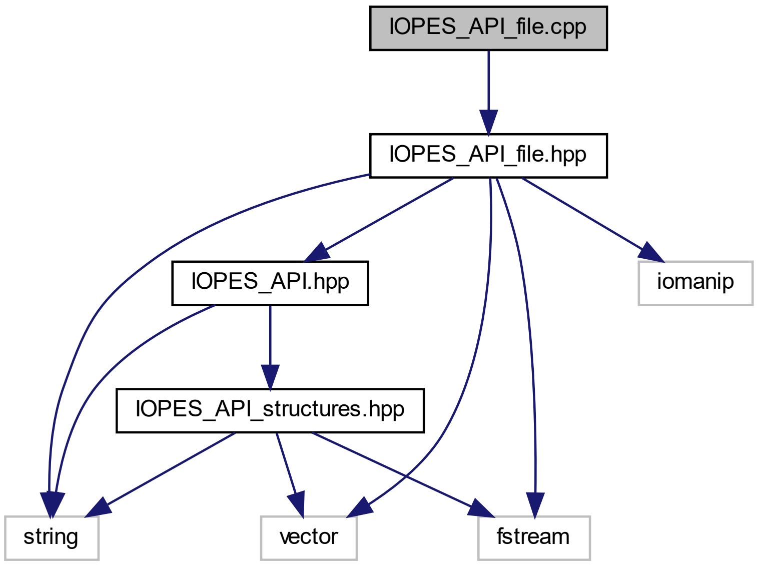
Implementation of the IOPES API using the curl library.

7.5 IOPES_API_file.cpp File Reference

Implementation file for IOPES_API_file.hpp.

```
#include "IOPES_API_file.hpp"
```

Include dependency graph for IOPES_API_file.cpp:



7.5.1 Detailed Description

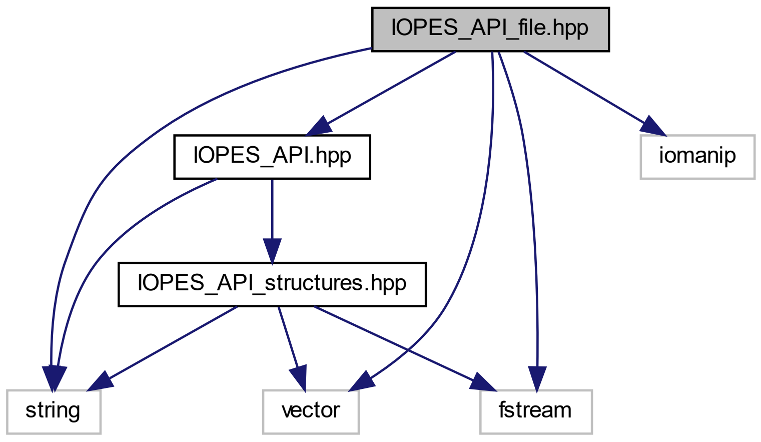
Implementation file for IOPES_API_file.hpp.

7.6 IOPES_API_file.hpp File Reference

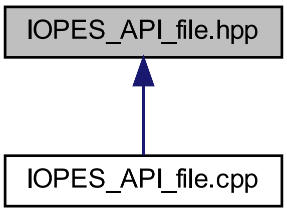
Implementation of the IOPES API using files as the backend. This class will must be used for testing purposes ONLY.

```
#include "IOPES_API.hpp"
#include <fstream>
#include <iomanip>
#include <string>
#include <vector>
```

Include dependency graph for IOPES_API_file.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class IOPES_API_file

  Implementation of the IOPES API using files as the backend. This class will must be used for testing purposes ONLY.

7.6.1 Detailed Description

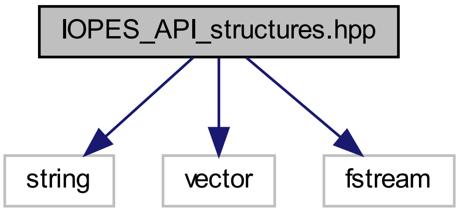
Implementation of the IOPES API using files as the backend. This class will must be used for testing purposes ONLY.

7.7 IOPES_API_structures.hpp File Reference

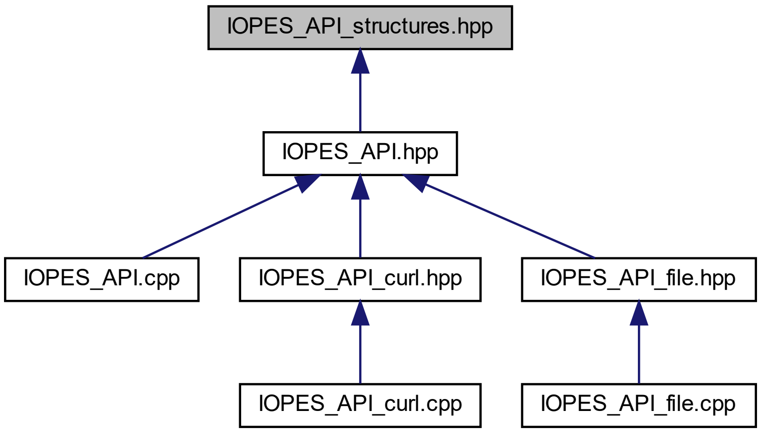
Header file defining useful data structures for the C++ implementation of the IOPES API using the libcurl library (IOPES_API_curl.hpp).

```
#include <string>
#include <vector>
#include <fstream>
```

Include dependency graph for IOPES_API_structures.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct IOPES_API_response

  Structure used to retrieve responses after calling an entry point.
- struct IOPES_API_tracker

  Structure defining a tracker (position)

7.7.1 Detailed Description

Header file defining useful data structures for the C++ implementation of the IOPES API using the libcurl library (IOPES_API_curl.hpp).

Index

IOPES

Indoor-Outdoor Positioning
for Emergency Staff