Tobii Pro Glasses 2 API
Developer's Guide

tobii pro

Developer's Guide Tobii Pro Glasses 2 API

Version 1.12.2
10/2016

Please check Tobii web site www.tobiipro.com for updated versions of this document.

Table of Contents

1 Introduction

1.1 Overview

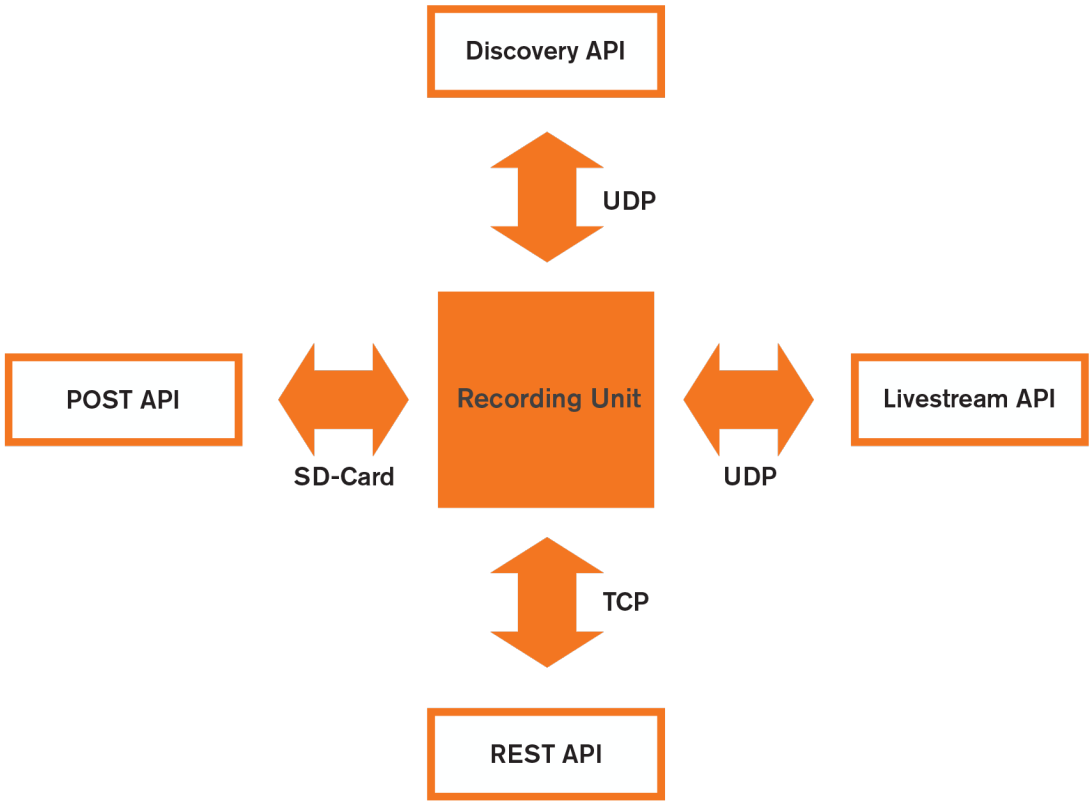
The Tobii Pro Glasses 2 eye tracking system is designed to be used for research purposes with adult participants and it includes the lightweight Tobii Pro Glasses Head Unit, a wearable Tobii Pro Glasses Recording Unit and Tobii Glasses Controller Software (running on a Windows 8 or later Pro tablet or any Windows 7 or later computer) or Tobii Pro Glasses 2 API running on any device or computer. The tablet/computer may or may not be included in the shipment depending on what package was purchased.

To record eye tracking data, the Tobii Pro Glasses Head Unit must be fitted onto the test participant's head (similar to a standard pair of glasses). The system must then be calibrated separately for each participant. In the calibration process the test participant is asked to look at a Calibration Card held in-front of the participant for a few seconds. The researcher then starts the recording from Tobii Glasses Controller Software or Tobii Pro Glasses 2 API running on any device or computer. After the session, the researcher stops the recording and removes the Tobii Pro Glasses Head Unit from the test participant. All interactions with the eye tracker (adding participants to test, initiating calibration, starting/stopping recordings etc.) are done through Tobii Glasses Controller Software or Tobii Pro Glasses 2 API. The Tobii Glasses Controller Software or Tobii Pro Glasses 2 API also enable the researcher to view/hear the eye tracking session both in real-time (streamed through a wireless or wired connection) and after the recording. When viewing a recording, you can hear what was recorded on the integrated microphone of the Tobii Pro Glasses 2 Head unit, the participant's gaze point also appears as a colored marker on the scene camera video from the Full HD camera integrated in the Tobii Pro Glasses 2 Head Unit (This is how it is presented in Tobii Glasses Controller Software, it is available through the API also).

For any eye tracking analysis beyond looking at the eye tracking replay (as described above), recorded data may be transferred to a computer running Tobii Glasses Analysis Software, alternatively you can see *Appendix C File Specifications, page 23* for information on the file format structure and use the data in another way. Tobii Glasses Analysis Software runs on Windows computers and must be purchased separately.

The Tobii Pro Glasses Recording Unit has four API interfaces.

- POST API:
  This API is stored on a SD-card and contains all scene camera and gaze data stored during recording and calibration.
- REST API:
  Tis API is used to control the Tobii Pro Glasses Recording Unit, e.g. to create projects, start and stop calibrations and recordings, but it can also be used to retrieve Tobii Pro Glasses Recording Unit status and information of the Tobii Pro Glasses Recording Unit and its head unit.
- Livestream API:
  This API can be used to get live data and video in real time.
- Discovery API:
  This API can be used to discover a Tobii Pro Glasses Recording Unit over the network.



1.2 Disclaimer of any warranty

Tobii reserves the right to change the structure or content of all files described in this document at any time without informing any external parties. Tobii leaves no warranty expressed or implied of any kind on products developed using the information provided in this document.

2 Concepts

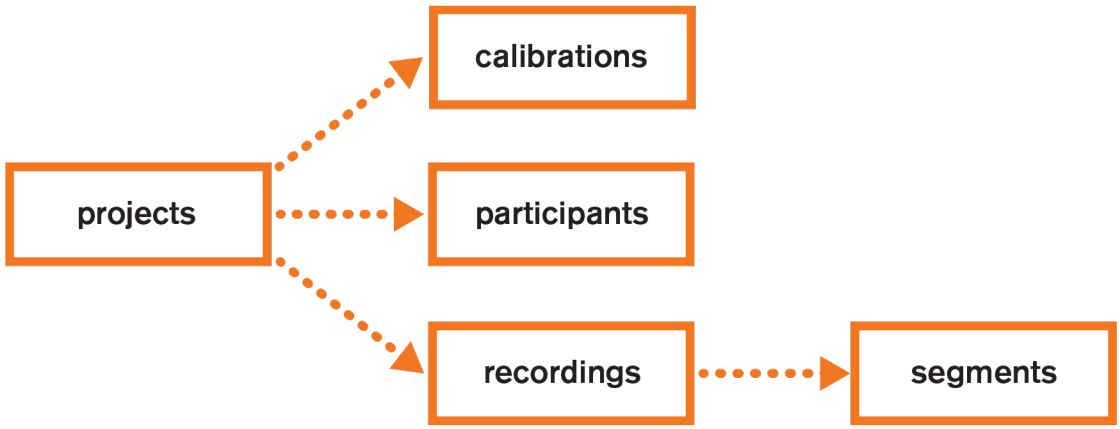
2.1 JSON objects

Data is stored in the form of JSON objects. During recording the objects can be read through the REST interface. In a finished recording they are stored as files.

Example:

```
{
    "pr_id": "4egezsr",
    "pr_info": {
        "CreationDate": "03/04/2015 08:32:23",
        "CustomId": "13ba5949-9ddf-4f9b-bf10-92e7711c18a4",
        "Name": "RecordingsExploratory - Release"
    },
     "pr_created": 2015-04-15T08:24:36+0000
...
}
```

Each entity (project, participant, calibration and recording) has a generated id that is guaranteed to be unique among all entities on the same SD-card. Segments do not have a unique identity. They are uniquely identified by their recording and the order in which they appear. Each entity also has a generic way of storing client specific data as a property object. In the example above, the project-object has a pr_info-property that stores three client specific values, CreationDate, CustomerId and Name.



2.2 Project

A project contains participants, recordings and calibrations.

The properties of the project can be accessed through the *REST API* and the data is stored in the file project.json.

2.3 Participant

A participant represent a unique person that uses the glasses.

A participant is required to do a calibration and a recording.

A participant can perform multiple calibrations and recordings.

A participant belongs to a single project.

A participant references a project and the participants most recently performed calibration.

The properties of the participant can be accessed through the *REST API* and the data is stored in the file participant.json.

2.4 Calibration

A calibration adapts the system to track a participant.

The calibration data gets copied to the current segment.

A calibration is never modified.

A calibration can be accessed through the *REST API* and the data is stored in the file calibration.json.

Tobii Pro Glasses Recording Unit is able to perform eyetracking without any calibration by using a default calibration (not recommended), but the accuracy is significantly worse for most participants.

A visual assessment of tracking quality is possible by instructing the participant to look at a calibration marker and using the live-view function of Tobii Glasses Controller. If the gaze is rendered on the calibration marker, the participant has a good calibration.

After a calibration has been performed, it is active until you complete a recording. If the calibration fails, the eye tracker will revert to the default calibration.

Known issue: It is currently not possible to reuse a calibration.

2.5 Recording

A recording belongs to a single project.

Each recording contains a number of segments. Whenever a recording is started the participant information is copied to the recording.

The properties of the recording can be accessed through the *REST API* and the data is stored in the file recording.json.

2.6 Segment

A segment is a part of a recording. When starting a new recording, an initial segment will be created. Additional segments can be created for a number of reasons;

- The recording is paused and resumed
- the video file becomes so large that it has to be split (2 GB, ~55 minutes @ 5 Mbit)
- Other internal reasons

Each segment contains all collected data (video, audio, eye tracking and MEMS (accelerometer and gyroscope) data).

Each segment has a reference to a calibration and a recording through the file copied during the calibration step (and whenever a new segment is created).

The properties of the segment can be accessed through the *REST API* and the data is stored in the file segment.json.

MEMS data is not available before firmware version 1.1.0.

2.7 Internal clock

If the Tobii Pro Glasses Recording Unit is able to connect with a ntp-server, it will sync the local hardware clock to the ntp-server clock. This clock is backed up by a battery that lasts for about 2 weeks. If the battery has been reset, the Tobii Pro Glasses Recording Unit will sync with the first date-information that is sent in a discovery-ping from a client.

3 Network

Please see the discover_glasses_on_network.py file located within the Tobii Pro Glasses 2 API for an example with notations.

3.1 Network address auto configuration

There are two different modes that the Tobii Pro Glasses Recording Unit may operate in depending on the network setup, these are described in the following section.

3.2 IPv4

The Tobii Pro Glasses Recording Unit will attempt to acquire an IPv4 address through DHCP on the wired network.

When the Tobii Pro Glasses Recording Unit operates in WLAN host mode the IPv4 address is 192.168.71.50.

If the device is connected directly to the machine (peer to peer network), the IPv4 address on the Ethernet interface is 0.0.0.0 and cannot be used.

There is no broadcast on IPv4.

3.3 IPv6

The Tobii Pro Glasses Recording Unit will assign itself a link-local fe80::.. IPv6 address that is reachable by all locally connected devices with no need for configuration. IPv6 address is a local address that will always remain the same.

The broadcast is only sent on IPv6.

3.4 Discovery

3.4.1 Discovery-broadcast

“Discovery” is the protocol by which the Tobii Pro Glasses Recording Unit can be found on the network. The Tobii Pro Glasses Recording Unit broadcasts a message with regular intervals over IPv6/UDP, on port 13006. This message contains the identity (serial number) of the Tobii Pro Glasses Recording Unit and also the firmware version, name and IP4-address if available.

To find a specific Tobii Pro Glasses Recording Unit in a slightly quicker way it is possible to send a discovery-ping on UDP port 13006.

{ ”type”:”discover”, ”date”:”2014-09-24T12:13:14Z”}

The date is optional.

3.4.2 Discovery-response

The response is sent as a broadcast on port 13006 and with the format:

{ ”type”:”identity”, ”class”:”glasses2”, ”name”:”MyTobiiG2”, ”version”:”1.0.4-246-elefantora-g35a0b15”, ”id”:”TG02B-080104031171”, ”port”:80, ”interface”:”eth0”, ”ipv4”:”10.46.16.4”}

The id is unique for a unit and will always be the same as the serial# printed on the sticker on the Tobii Pro Glasses Recording Unit.

The name can be changed through the *REST API*.

IPv4 address will be the address of the network interface that received the discovery-message.

“port” is the port that should be used to access the *REST API*.

“interface” can be either “eth0” (if the Tobii Pro Glasses Recording Unit is connected via LAN cable) or “wlan0” (if the Tobii Pro Glasses Recording Unit is connected via wireless network).

The IPv6 address has to be retrieved from the network response and is not part of the discovery-response JSON.

When changing the name of the Tobii Pro Glasses Recording Unit using the procedure described in section 7 *REST API, page 14*, it will not affect the response from discovery, until the discovery is restarted (when the Tobii Pro Glasses Recording Unit is rebooted).

4 Livestream API via UDP

The Tobii Pro Glasses Recording Unit (recording unit) can transmit both a live videostream from the scene camera and a live datastream from the eye tracker via UDP. The livestreams can be requested any time the Tobii Pro Glasses Head Unit is connected, this does not require an ongoing recording. They can be initiated and terminated before, during or after a recording.

Please see the livestream_data_and_video.py file located within the Tobii Pro Glasses 2 API for an example with notations.

4.1 Keep-alive messages

Live streaming is started by sending keep-alive messages to the live ctrl port (defined by the **sys_livectrl_port** property in the system-info JSON, see the procedure in section7 *REST API, page 14*). The messages should be sent regularly, at an interval specified by the **sys_livectrl_ka** property in the system-info JSON. If the message has not been received for three intervals, the Tobii Pro Glasses Recording Unit will stop transmitting to the client.

There is one keep-alive message for the livevideo stream and one message for the live datastream. Both keep-alive messages are JSON objects containing three properties, **op** (operation), **type**, and **key,** as illustrated below.

{ ”op”: ”start”, ”type”: “live.data.unicast”, ”key”: ”62b3a246-2e4c-46ab-8082-f3ed7094e553”}

Where the **op** property can have the value *start* and *stop*. To stop a stream explicitly, send a keep-alive message with the **op** property value set to “stop”. The **type** property defines which live stream will be transmitted, value *live.video.unicast* for the video stream, and *live.data.unicast* for the data stream. The **key** property can be any string, but should be unique for the client since it is used by the recording unit to handle reference counting of the active data streams.

The two keep-alive messages need to be transmitted via different sockets.

Currently both keep-alive messages need to be transmitted in order to receive any live video. If only the keep-alive message for data is transmitted the gaze data will be invalid. Hence, both keep-alive messages needs to be sent in order to receive good data.

4.2 Live video (Scene camera)

The live video from the scene camera (Full HD, 1920x1080, 25 fps), is encoded into the h.264 format with keyframes every 16 frames at ~5Mbit and the audio from the Tobii Pro Glasses Head Unit microphone (24 kHz, mono) is encoded into the mp3 format. It is transmitted via UDP as mpeg-ts (MPEG transport stream) packets (188 bytes each).

The live videostream is sent to the same socket and port as its corresponding received keep-alive message.

4.3 Live data (Eyetracker and MEMS data)

In the live datastream JSON; messages with information from the eyetracker and the MEMS-sensor are transmitted. All messages in this stream contains a status property **s** and a time stamp property **ts**. If the status property **s** has the value *0*, this means that there are currently no errors, any non-zero value indicates some kind of problems with the data in that message. The **ts** property is a monotonic timestamp of the data in microseconds. Below the different messages are described. The **l** property is the latency, expressed in a number of microseconds from the time the image is received from the camera to the time it is queued for transmitting the live data stream. Live data elements in the live datastream is described in section B6.1

The live datastream is received via the same socket and port as its corresponding keep-alive message has been sent via.

MEMS data is not available before firmware version 1.1.0.

5 Synchronize Live Video and Gaze Data

In order to show live video with gaze overlay, it is important to synchronize the live videostream and live datastream. All JSON elements in the live datastream contain the property **ts**, which is the datastream timestamp value. The mpeg-ts live videostream is timestamped using **pts**, (see this Wikipedia article). The **ts** timestamp value is in microseconds. The **pts** timestamp value has a resolution of 90 kHz. The Pro Glasses Recording Unit sends a **pts** sync package (see *C6.1.8 PTS sync package, page 28*) that allows a client to translate the datastream time ts, to the videostream time pts. Use the offset between the **ts** property and the **pts** property in the sync package in order to sync the gaze data with the live videostream.

6 Synchronize Recorded Video and Gaze Data

The gaze data stream contains a sync packet between **ts** time and **vts** time (see *5 Synchronize Live Video and Gaze Data, page 12*). The video file starts with a key-frame.

For firmware 1.6.9 and earlier, the video file does not start with a key-frame. The consequence of this is that the first frames might not be possible to show.

For firmware 1.0.3 and earlier, when syncing both live data and video file, it is required to subtract an additional 120 ms (on top of the ts-vts/pts offset) from the gaze data timestamp to get a matching video timestamp. From firmware 1.1.0 and later, no offset is required.

7 REST API

This section contains information and a description of the REST API. The API is self-documenting and described at http://<address-of-RU>/services, see *Appendix E REST API Services*

Please see the calibrate_and_record.py file located within the Tobii Pro Glasses 2 API for an example with notations.

REST API is using HTTP as described in RFC 2068 (see http://ietf.org/rfc/rfc2068.txt)

To get a list of available REST API calls:

http://<address-of-RU>/services

The base URL used for the REST API is prefixed with /api:

http://<address-of-RU>/api/<resource>

7.1 CRUD operations

Full CRUD (Create, Read, Update, Delete) are supported unless otherwise stated.

The below are just format examples and do not represent the actual data.

HTTP methods match to CRUD operations as:

Request	CRUD	Example	Description
POST	Create/ Update	POST /api/projects HTTP/1.1 Content-Type: application/json Content-Length: 48 {"pr_info":{"name":"my new project","xid":"19"}} **Returns** HTTP/1.1 201 Created Connection: Keep-Alive Content-Length: 137 Pragma: no-cache Cache-Control: no-cache Content-Type: application/json; charset=utf-8 Location: /api/projects/73z7vtv Date: Mon, 13 Apr 2015 11:10:27 GMT {"pr_id":"73z7vtv","pr_info":{"name":"my new project","-xid":"19"},"pr_created":"2015-04-13T11:10:27 +0000","uri":"/api/projects/73z7vtv"}	Create new project. The application should use the provided URI in the *Location:* header for any subsequent operations on project. If an info object already exists, it will be overwritten when updating using POST.

PUT	Update	PUT /api/projects/73z7vtv HTTP/1.1 Content-Type: application/json Content-Length: 31 {"pr_info":{"name":"Project1"}} **Returns** HTTP/1.1 200 OK Connection: Keep-Alive Content-Length: 120 Pragma: no-cache Cache-Control: no-cache Content-Type: application/json; charset=utf-8 Date: Mon, 13 Apr 2015 11:13:50 GMT {"pr_id":"73z7vtv","pr_info":{"name":"Project1"},"pr_created":"2015-04-13T11:10:27+0000","uri":"/api/projects/73z7vtv"}	Update project definition.
GET	Read	GET /api/projects/73z7vtv HTTP/1.1 **Returns** HTTP/1.1 200 OK Connection: Keep-Alive Content-Length: 120 Pragma: no-cache Cache-Control: no-cache Content-Type: application/json; charset=utf-8 Date: Mon, 13 Apr 2015 11:14:32 GMT {"pr_id":"73z7vtv","pr_info":{"name":"Project1"},"pr_created":"2015-04-13T11:10:27+0000","uri":"/api/projects/73z7vtv"}	Return project definition
DELETE	Delete	DELETE /api/projects/73z7vtv HTTP/1.1 **Returns** HTTP/1.1 204 No Content Connection: Keep-Alive Content-Length: 1 Pragma: no-cache Cache-Control: no-cache Content-Type: application/json; charset=utf-8 Date: Mon, 13 Apr 2015 11:15:34 GMT	Delete project

7.2 Projects

For more information please see *C2 Project Folder, page 23*

7.2.1 Creating a new project

POST /api/projects HTTP/1.1

7.2.2 Updating a project

```
POST /api/projects/<project id> HTTP/1.1
```

Use a standard POST request to update the fields.

7.3 Participants

For more information please see *C4 Participant Folder, page 24*

7.3.1 Creating a new participant

```
POST /api/participants HTTP/1.1
```

pa_project is a required field.

7.3.2 Updating a participant

```
POST /api/participants/<participant id> HTTP/1.1
```

Use a standard POST request to update the fields.

7.4 Calibrations

For more information please see *C3 Calibration Folder, page 23*

7.4.1 Creating a new calibration

```
POST /api/calibrations HTTP/1.1
```

Required fields are ca_participant and ca_type.

ca_type should be set to default.

7.4.2 Updating a calibration

```
POST /api/calibrations/<calibration id> HTTP/1.1
```

Use a standard POST request to update the fields.

7.4.3 Sending a calibration marker

This information is sent during calibration whenever the image recognition process detects the calibration marker in the scenecamera image or when a virtual marker is received via *REST API*. The **marker2d** property values have the same unit as the **gp** property in the Gaze position message and the **marker3d** values have the same unit as the **gp3** property in the Gaze position 3D message.

```
{
    "ts":1113243866,
    "marker2d":[0.5678, 0.2311],
    "marker3d":[31.322,27.654,22.442],
    "s":0
}
```

```
POST /api/calibrations/<ca_id>/marker/<x>/<y>/<z>
```

This function is used only in special cases when there is a need to specify the position of the calibration marker position. Use a standard POST request to send a calibration marker. <x>, <y> and <z> are float values for marker coordinates. These coordinates are in the scene camera coordinate system and measured in [mm] (See *Appendix A Gaze Direction Coordinate system, page 21*).

Only available from firmware version 1.1.0 onwards.

7.5 Recordings

For more information please see *C5 Recording Folder, page 24*

7.5.1 Creating a new recording

```
POST /api/recordings HTTP/1.1
```

ec_participant is a required field.

7.5.2 Updating a recording

```
POST /api/recordings/<recording id> HTTP/1.1
```

Use a standard POST request to update the fields.

7.6 Configuration

7.6.1 Updating the system config.

```
POST /api/system/conf HTTP/1.1
```

Use a standard POST request to update the fields.

7.6.2 Changing eye tracking frequency

In api/system/status all available frequencies for the currently connected Head Unit are in a list of integers in sys_et.frequencies (see *Appendix D System Status*).
In api/system/conf the currently selected frequency is visible in the sys_et_freq object. A user can write any frequency available in sys_et. frequencies to the sys_et_freq object to change frequency. Changing frequency is not allowed during calibration and/or during recording.

Changing eye tracking frequency is not available before firmware version 1.12.2.

7.7 Sending external events/sync signals

```
POST /api/events HTTP 1.1
```

Events can be sent via a POST request. Metadata about any external events/signals can posted as a json object (see example below). Data will end up in livedata.json file in a recording; without an on going recording this data will not be stored anywhere.

Sample Json data that can be sent:

```
{
        'ets': 67967988,
        'type': 'BigEvent',
        'tag': 'some data which could be one-line formatted json, xml or whatever'
}
```

Where 'ets' is external timestamp, 'type' and 'tag' are strings that can be anything the user wants them to be but limited by 1024 characters in total. User must send a string value for 'type' but 'tag' is optional. If no 'ets' is sent then it will be zero. This information will be stored to the livedata json file (see *C6.1.11 API-sync package*).

7.8 Wlan Settings

The interface to control wlan is located at /api/system/wlan, with the following format:

```
{
    "sys_wlan_mode": "client",
    "sys_wlan_enabled": "true"
    "sys_wlan_ap_ssid": "TG02B-080105006461",
    "sys_wlan_ap_key": "TobiiGlasses2",
    "sys_wlan_ap_channel": 6,
    "sys_wlan_client_ssid": "GuestWlan",
    "sys_wlan_client_key": "VeryVerySecret"
    "uri:" "/api/system/wlan"
}
```

Wlan settings are not available before firmware version 1.5.2.

This contains the Wlan settings for the RU. It can be read and updated with standard REST requests. After each request that modifies the settings, the wlan will be restarted with the new settings. The call will return immediately, but it may take some time before the settings actually apply to the unit. Progress can be observed by reading api/system/status, described in *Appendix D System Status*.

sys_wlan_mode

Value Type : String

Description: The function mode of the Wlan function. Possible values are {ap, client}.

- In “ap” mode, the unit will act as an access point, using the settings sys_wlan_ap_ssid, sys_wlan_ap_key, and sys_wlan_ap_channel.
- In “client” mode, the RU will attempt to connect to an existing network, using the settings sys_wlan_client_ssid and sys_wlan_client_key.

sys_wlan_enabled

Value Type : Boolean

Description: Indicates whether the Wlan function is enabled or not.

sys_wlan_ap_ssid

Value Type : String

Description: The SSID of the unit used in access point mode (1–31 chars).

sys_wlan_ap_key

Value Type : String

Description: WPA2 passphrase (8–63 chars).

sys_wlan_ap_ssid

Value Type : String

Description: The SSID of the unit used in access point mode (1–31 chars).

sys_wlan_ap_channel

Value Type : Int

Description: The network channel (1–11).

sys_wlan_client_ssid

Value Type : String

Description: The SSID to be used in client mode connecting to an existing network (1–31 chars).

sys_wlan_client_key

Value Type : String

Description: WPA2 passphrase for connecting to an existing network in client mode. (8–63 chars).

7.8.1 Resetting Wlan settings

This operation /api/system/wlan/reset will reset the Wlan settings to the factory defaults.

7.8.2 Scanning for networks

This operation /api/system/wlan/scan_networks will scan for available networks.

This operation is only available in client mode.

7.9 Custom property objects

It is possible to store custom properties in projects, participants, calibrations, recordings and system/conf. There is an xyz_info field that may be used to store custom information. The custom property fields are as follows:

pr_info for projects,

pa_info for participants,

ca_info for calibrations,

rec_info for recordings,

sys_info for system/conf.

The information is set using a standard CRUD operation (POST or PUT).

```
POST /api/projects/<project id> HTTP/1.1
```

Example:

```
POST /api/projects/z3wfmi5 HTTP/1.1
Content-Type: application/json
Content-Length: 42
{"pr_info":{"name":"Project1","xid":"01"}}
```

7.10 Non CRUD operations

- /api/system/eject - syncs data and unmounts the sd-card. If a file is locked, the unmount will fail. If the unmount is successful, the green SD LED should turn off.
- /api/identify - flashes the LEDs of the device 3 times to make it easier to identify the correct unit when there are multiple devices on the network.
- /api/system/status – Contains the current status of the unit.
- /api/system/wlan – Contains the unit's WLAN settings.

7.11 Persistent status push

The "/api.../status" interface returns the current state of the resource and then closes the connection. To provide a live feed of status updates, without the need to poll the status URL, a query parameter "?persistent" may be added, this will force the connection to be kept alive and automatically push any status updates to the client. The current status will be periodically updated and sent to the client.

E.g.:

```
GET /api/system/status?persistent HTTP/1.1
..
Returns
HTTP/1.1 200 OK
Transfer-Encoding: chunked
{ ... }
-- time passes
{ .... }
...
```

7.12 Error reporting

If an error is encountered while processing an API request an error reply is returned with a HTTP error code, the content will be a JSON object with a predefined format:

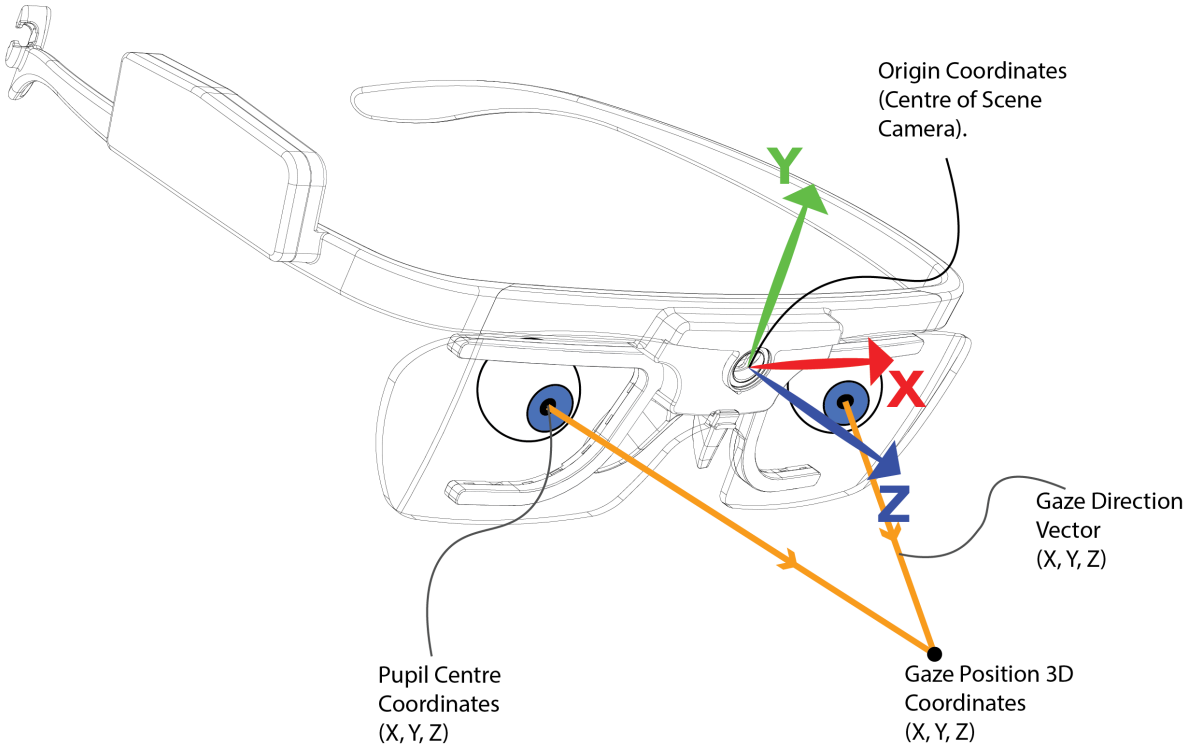
Example — When a required field (rec_participant) is missing in the POST request.

```
POST /api/recordings HTTP/1.1
Content-Type: application/json
Content-Length: 30
{"rec_info":{"name":"lalala"}}
HTTP/1.1 400 Bad Request
Connection: Keep-Alive
Content-Length: 96
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/json; charset=utf-8
Date: Mon, 13 Apr 2015 10:31:49 GMT
{"code":"generic.invalidinput","reason":"Failed to parse JSON: Missing field: rec_participant"}
```

Required fields are checked for errors.

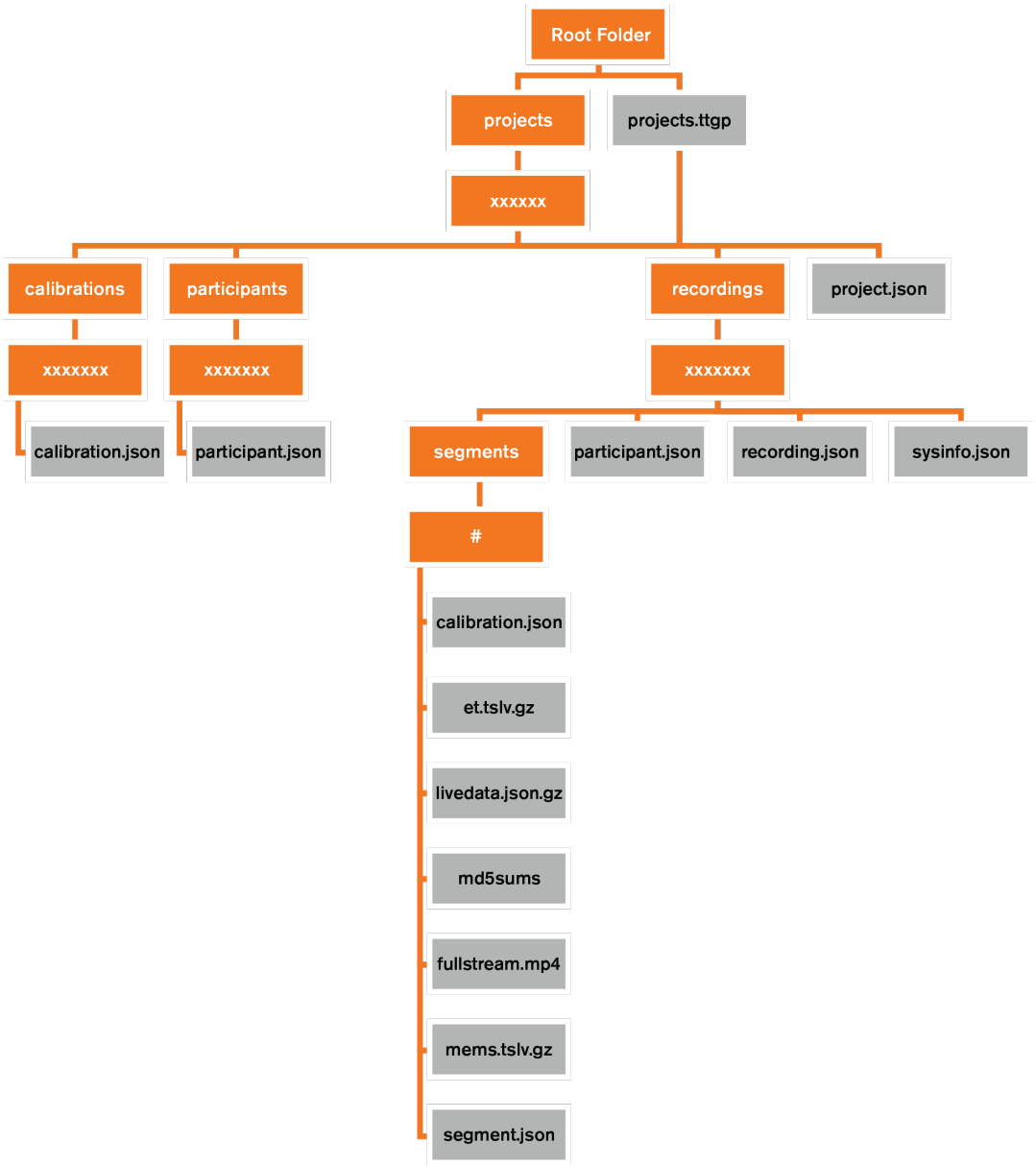
Misspelling the name of a field that is not required will not be reported as an error and the field and data are discarded!

Appendix A Gaze Direction Coordinate system



Appendix B File Structure

The figure below illustrates the file structure on the SD card saved by the Tobii Glasses 2 Recording Unit.



Appendix C File Specifications

C1 Root Folder

C1.1 projects.ttgp

Empty file for internal project structure purposes.

C2 Project Folder

C2.1 project.json

Holding per Project Meta data, such as name and date created.


```
{
    "pr_id": "k3s6eeu",
    "pr_info": {
        "CreationDate": "07/01/2016 09:23:19",
        "EagleId": "d34fde7c-6e88-4a6e-b2f9-4c1742dbfa3e",
        "Name": "Study001"
    },
    "pr_created": "2016-07-01T09:25:23+0000"
}
```


C3 Calibration Folder

C3.1 calibration.json

Contains calibration data.


```
{
    "ca_id": "lpc6fn3",
    "ca_info": {

    },
    "ca_participant": "guztqrv",
    "ca_state": "failed",
    "ca_data": "TKM7SQE2TGEUAAAAAAAAAAAAACNJT6KATKMYSQAAAAAAAAAAAAAAA",
    "ca_error": "Calibration timed out due to: lacking left eye features",
    "ca_error_code": 211,
    "ca_error_info": "2MAAAAAAAAAAA",
    "ca_type": "default",
    "ca_created": "2016-10-21T12:30:39+0000",
    "ca_project": "wgbxrmd"
}
```

C4 Participant Folder

C4.1 participant.json

Contains participant information such as ID and name:


```
{
    "pa_id": "25iquhx",
    "pa_info": {
        "EagleId": "75b28665-c973-407f-bcdf-71d4f14796d9",
        "Name": "Jonas Z",
        "Notes": ""
    },
    "pa_project": "k3s6eeu",
    "pa_calibration": "5me6abi",
    "pa_created": "2016-07-01T11:57:22+0000"
}
```


C5 Recording Folder

C5.1 participant.json

Contains participant information for the given recording.


```
{
    "pa_id": "25iquhx",
    "pa_info": {
        "EagleId": "75b28665-c973-407f-bcdf-71d4f14796d9",
        "Name": "Jonas Z",
        "Notes": ""
    },
    "pa_project": "k3s6eeu",
    "pa_calibration": "5me6abi",
    "pa_created": "2016-07-01T11:57:22+0000"
}
```

C5.2 recording.json

Contains information about the recording and the tracksphere.


```
{
    "rec_id": "f3fml27",
    "rec_info": {        "EagleId": "78bcdc89-002a-4c0e-8406-e2cef8879c8b",
        "Name": "Recording004",
        "Notes": ""    },
    "rec_participant": "wcvqpo2",
    "rec_project": "k3s6eeu",
    "rec_state": "done",
    "rec_segments": 1,
    "rec_length": 243,
    "rec_calibration": "37k4jxj",
    "rec_created": "2016-07-01T11:21:33+0000",
    "rec_et_samples": 24366,
    "rec_et_valid_samples": 20592,
    "ts_right_x": -32.5,
    "ts_right_y": -27.0,
    "ts_right_z": -19.0,
    "ts_left_x": 32.5,
    "ts_left_y": -27.0,
    "ts_left_z": -19.0,
    "ts_green_limit_radius": 10.0,
    "ts_yellow_limit_radius": 12.5
}
```


C5.3 sysinfo.json

Contains system information for the given recording.


```
{    "servicemanager_version": "1.12.2-quornburgare",
    "fpga_v_maj": 0,
    "fpga_v_min": 0,
    "fpga_v_rel": 62,
    "fpga_variant": "normal",
    "board_type": "DVT Board",
    "hu_serial": "TG02G-010105732624",
    "ru_serial": "TG02B-080104030181"
}
```


C6 Segments Folder

C6.1 livedata.json.gz

Compressed (gzip) JSON file containing eye tracking data and gaze vectors. All messages in this stream contain a status indicator “s”. Zero means everything is OK, any non-zero value indicates some kind of problem with the data. It is important to note that the entire *livedata.json* file itself is not a properly formatted JSON file. Instead each row making the file is a JSON object. Hence, the JSON data in *livedata.json* must be read and parsed row by row.

C6.1.1 Pupil Center

The property **pc** is specified in 3D coordinates with origo in the scenecam. This can be used to compare the eye position with the tracksphere from the status report in the *7 REST API, page 14*. The value is sent separately for each eye and the coordinates are in mm.


```
{
    "ts":1113243866,
    "eye":"right",
    "pc":[-31.322,-27.654,-22.442],
    "s":0
}
```


C6.1.2 Pupil Diameter

The pupil diameter is measured in mm and sent separately for each eye


```
{
    "ts":1113243866,
    "eye":"right",
    "pd":2.674
    "s":0,
}
```


C6.1.3 Gaze Direction

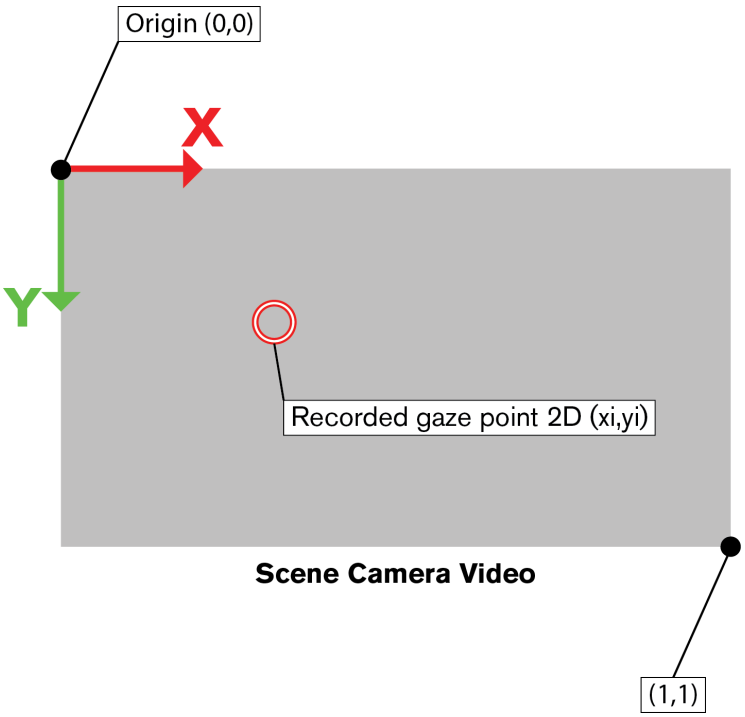
The gaze direction is a unit vector with origo in the pupil center


```
{
    "ts":1113243866,
    "eye":"right",
    "gd":[0.646,0.583,0.494],
    "s":0
}
```

C6.1.4 Gaze Position

The gaze position is the position on the scene camera image where the gaze will be projected. Top left corner is (0,0), bottom right corner is (1,1).


```
{
        "ts":1114210113
        "gp":[0.541,0.172],
        "s":0
        "l":17013
}
```

C6.1.5 Gaze Position 3d

GazePosition3d is the 3D position, in mm, relative to the scene camera where the gaze is focused.


```
{
    "ts":1114210113,
    "gp3":[0.454,-1.149,4.125],
    "s":0
}
```


C6.1.6 MEMS gyroscope info

The gyroscope data indicates the rotation of the glasses. The gyroscope data has the unit degrees per second [°/s]. It is activated from firmware version 1.1.0.


```
{
    "ts":1114210113,
    "gy":[0.454,-1.149,4.125],
    "s":0
}
```

C6.1.7 MEMS accelerometer info

The accelerometer data indicates the rotation of the glasses. The accelerometer data has the unit meter per second squared [m/s²]. When the glasses are stationary, the value of the **ac** property will be approximately [0, -9,82, 0]. It is activated from firmware version 1.1.0.

```
{
    "ts":1114210113,
    "ac":[0.454,-1.149,4.125],
    "s":0
}
```

C6.1.8 PTS sync package

The pts sync package is used to get the offset between the PTS time in the video to the TS time that is used in TSLV and JSON-files. “pv” is the “pipeline version” and will change every time the pipeline is restarted for some reason (and the PTS values have to be reset). The **pts** property value has a resolution of 90 kHz in relation to the TS time since the video pipeline started.

```
{
    ”ts”:1114188324,
    ”pts”:1201701,
    ”pv”:1
    ”s”:0,
}
```

C6.1.9 VTS sync package

The **vts** sync package is used to get the offset between the video time in the mp4-file and the TS-time that is used in TSLV and JSON-files. There will always be one **vts** package for the first frame (vts=0), and one **vts** package for the first keyframe in the video (could be frame 0, but should be one of the 16 first frames). After this, there will be a **vts** package approximately once per minute.

```
{
    ”ts”:1114188324,
    ”vts”:0,
    ”s”:0,
}
```

C6.1.10 Sync-port signal package

This sync package contains a log of signals sent. The example below "dir" indicates the direction, which can be "in" or "out". "sig" indicates a signal where 1 means a 3.3 V signal on the port and 0 means a 0.0 V signal.


```
{
    "ts":16956062532,
    "s":0,
    "dir":"out",
    "sig":1
}
```


C6.1.11 API-sync package

This package contains a log of events/signals sent over REST api. The example log data below consists of "ets" "tag" and "type" will match what is posted using the external event API.


```
{
        "ts":1926483984,
        "s":0,
        "ets":67967988,
        "type":"BigEvent",
        "tag":"some data which could be one-line formatted json, xml or whatever"
}
```


C6.2 calibration.json

Contains calibration information.


```
{
    "ca_id": "lpc6fn3",
    "ca_info": {

    },
    "ca_participant": "guztqrv",
    "ca_state": "failed",
    "ca_data": "TKM7SQE2TGEUAAAAAAAAAAAAACNJT6KATKMYSQAAAAAAAAAAAAAA",
    "ca_error": "Calibration timed out due to: lacking left eye features",
    "ca_error_code": 211,
    "ca_error_info": "2MAAAAAAAAAAA",
    "ca_type": "default",
    "ca_created": "2016-10-21T12:30:39+0000",
    "ca_project": "wgbxrmd"
}
```


C6.3 segment.json

Contains recording segment information such as id, duration, start and stop:


```
{
    "seg_id": 1,
    "seg_length": 243,
    "seg_length_us": 243240938,
    "seg_calibrating": true,
    "seg_calibrated": true,
    "seg_t_start": "2016-07-01T11:23:56+0000",
    "seg_t_stop": "2016-07-01T11:27:59+0000",
    "seg_created": "2016-07-01T11:23:56+0000",
    "seg_end_reason": "api"
}
```

C6.4 mems.tslv.gz

Compressed (gzip) binary file that will hold MEMS data.

MEMS data is not available before firmware version 1.1.0.

C6.5 et.tslv.gz

Compressed (gzip) binary file holding essentially the same data as livedata.json.

Appendix D System Status


```
{
    "sys_name": "TG02B-080104030181",
    "sys_status": "ok",
    "sys_descr": "Glasses 2 controller",
    "sys_serial": "TG02B-080104030181",
    "sys_macaddr": "74fe48051d3d",
    "sys_hostname": "g2d",
    "sys_version": "1.12.2-quornburgare",
    "sys_api_version": "1.1.0",
    "sys_uptime": 2379,
    "sys_time": "2016-10-20T12:36:22+0000",
    "sys_upgrade": { },
    "sys_headunit": {
        "state": "up",
        "initialized": "false",
        "changed": "2016-10-20T12:36:08+0000",
        "fpga_v_maj": 0,
        "fpga_v_min": 0,
        "fpga_v_rel": 62,
        "fpga_variant": "normal"
    },
    "sys_wlan": {
        "status": "ap",
        "ssid": "TG02B-080104030181",
        "is_scanning": "false",
        "networks": [ ]
    },
    "sys_recording": { },
    "sys_etd_calibrated": "false",
    "sys_calibration": { },
    "sys_et": {
        "state": "idle",
        "changed": "2016-10-20T12:36:09+0000",
        "frequencies": [
            50,
            100
        ]
    },
    "sys_track_sphere": {
        "right_x": -32.5,
        "right_y": -27,
        "right_z": -19,
        "left_x": 32.5,
        "left_y": -27,
        "left_z": -19,
        "green_limit_radius": 10,
        "yellow_limit_radius": 12.5
    },
    "sys_mems": {
        "state": "idle",
        "changed": "2016-10-20T12:36:08+0000"
    },
    "sys_battery": {
        "status": "Discharging",
        "level": 64,
        "remaining_time": 3840
    },
    "sys_storage": {
        "type": "SD-card",
        "status": "available",
        "capacity": 32109494272,
        "remaining": 31413387264,
        "remaining_time": 47933,
        "volume_label": "",
        "volume_uuid": "4C0A-FDBC"
    },
    "sys_live_stream": {
        "live.video.unicast": 0,
        "live.data.unicast": 0
    }
}
```

D1 Field Descriptions

sys_name

Value Type : string

Description: The name of the system.

sys_descr

Value Type : string

Description: A description of the system.

sys_serial

Value Type : string

Description: The unique serial number of the unit.

sys_macaddr

Value Type : string

Description: The units mac address of the eth0 interface.

sys_hostname

Value Type : string

Description: The hostname of the system.

sys_version

Value Type : string

Description: The version of the system.

sys_api_version

Value Type : string

Description: The version of the REST API.

sys_sim_et

Value Type : string

Description: This is set when the eye tracker is set to simulation mode.

sys_sim_clb

Value Type : string

Description: This is set when using a dummy calibration.

sys_uptime

Value Type : int

Description: Uptime in seconds since boot.

sys_time

Value Type : string

Description: The system time in the format yyyy-mm-ddTHH:MM:SS+0000. All times are UTC.

Example: 2015-04-15T08:24:36+0000

sys_upgrade

Value Type : Object

Description: See *D2 Upgrade, page 33*.

sys_headunit

Value Type : Object

Description: See *D3 Headunit, page 34*.

sys_recording

Value Type : Object

Description: See*D4 Recording, page 34*.

sys_etd_calibrated

Value Type : boolean

Description: A boolean explaining if etd is calibrated. {true, false}

sys_ calibration

Value Type : Object

Description: See *D5 Calibration, page 34*.

sys_et

Value Type : Object

Description: See *D6 Et, page 35*.

sys_track_sphere

Value Type : Object

Description: See *D7 Tracksphere, page 35*.

sys_mems

Value Type : Object

Description: See *D8 Mems, page 35*.

sys_battery

Value Type : Object

Description: See *D9 Battery, page 36*.

sys_ storage

Value Type : Object

Description: See *D10 Storage, page 36*.

D2 Upgrade

packages

Value Type : Int

Description: The number of packages to upgrade.

progress

Value Type : Int

Description: The progress of upgrade process.

message

Value Type : String

Description: Message from the upgrade. This could be used to give a user feedback on the upgrade progress.

D3 Headunit

state

Value Type : string

Description: Possible values are {“disconnected”, “up”}

initialized

Value Type : boolean

Default Values: “false”

Description: Possible values are {“true”, “false”}

changed

Value Type : string

Description: The timestamp when the headunit last was changed or updated. In UTC.

Time format is yyyy-mm-ddTHH:MM:SS+0000.

fpga_v_maj

Value Type : Int

Description: Major version number of the head unit firmware.

fpga_v_min

Value Type : Int

Description: Minor version number of the head unit firmware.

fpga_v_rel

Value Type : Int

Description: Revision version number of the head unit firmware.

fpga_variant

Value Type : String

Description: The fpga variant. Possible values {normal, fallback}. If fallback is set then the FPGA has failed to read the normal boot area.

D4 Recording

rec_id

Value Type : Int

Description: The id of the current recording.

rec_state

Value Type : String

Description: The recording state. Possible values are: {init, starting, recording, pausing, paused, stopping, stopped, done, stale, failed}

D5 Calibration

ca_id

Value Type : Int

Description: The id of the current calibration.

ca_state

Value Type : String

Description: The calibration state. Possible values are: {calibrating, stale, uncalibrated}

D6 Et

state

Value Type : String

Description: The eye tracking state. Possible values are: {unavailable, idle, eyetracking, ...}

changed

Value Type : String

Description: The timestamp when the eye tracker state was changed or updated. In UTC.

Time format is yyyy-mm-ddTHH:MM:SS+0000.

frequencies

Value Type : Integer

Description: The available frequencies for currently connected Head Units.

D7 Tracksphere

right_x, right_y,right_z

Value Type : Int

Description: Optimal right eye x,y,z position.

left_x, left_y, left_z

Value Type : Int

Description: Optimal left eye x,y,z position.

green_limit_radius

Value Type : Int

Description: The radius for the indicator to turn green. Pupil distance from the center of the tracksphere where eyetracking is expected to use both eye cameras.

yellow_limit_radius

Value Type : Int

Description: The radius for the indicator to turn yellow. Pupil distance from the center of the tracksphere where eyetracking is expected to use at least one eye camera.

D8 Mems

state

Value Type : String

Description: The MEMS state. Possible values are: {idle, starting, running}

changed

Value Type : String

Description: The timestamp when the eye tracker state was changed or updated.

Time format is yyyy-mm-ddTHH:MM:SS+0000.

MEMS data is not available before firmware version 1.1.0.

D9 Battery

status

Value Type : String

Description: The battery status. Possible values are: {NoBattery, Discharging, Charging, Full}

level

Value Type : Int

Description: The battery charge level in %. The range is from 0 to 100.

remaining_time

Value Type : Int

Description: The estimated time before the battery is empty. The unit is in seconds.

D10 Storage

type

Value Type : String

Description: The storage type. Possible values are {SD-Card}

status

Value Type : String

Description: The status of the storage. Available values are {unknown, readonly, available}

capacity

Value Type : Unsigned long

Description: The max storage space of a device, in bytes.

remaining

Value Type : Unsigned long

Description: The available storage space on the device in bytes.

remaining_time

Value Type : Unsigned long

Description: The number of seconds of data that can be stored on the device.

volume_label

Value Type : String

Description: The volume label. This may be edited and changed from Windows.

volume_uuid

Value Type : String

Description: The volume id is an 8 character SDcard serial number of the card that is in the Tobii Pro Glasses 2 Recroding Unit.

Format is "volume_uuid": "9016-4EF8"

If the SD card is ejected, the value will be an empty string.

D11 Wlan

status

Value Type : String

Description: The status of the Wlan function. Possible values are {unavailable, disabled, enabling, ap, not_connected, connecting, connected}. Units not supporting Wlan will have the value “unavailable”.

ssid

Value Type : String

Description: The ssid of the unit when operating in AP mode (n/a if Wlan is disabled)..

is_scanning

Value Type : Boolean

Description: Indicates whether a network scan is in progress or not.

networks

Value Type : Array

Description: An array of JSON objects, one per visible network. JSON object fields are {ssid, encryption, signal-db}.

Appendix E REST API Services

```
/services
/api
/api/projects
/api/projects/<pr_id>
/api/projects/<pr_id>/participants
/api/projects/<pr_id>/recordings
/api/projects/<pr_id>/description
/api/participants
/api/participants/<pa_id>
/api/participants/<pa_id>/recordings
/api/participants/<pa_id>/calibrations
/api/participants/<pa_id>/description
/api/calibrations
/api/calibrations/<ca_id>
/api/calibrations/<ca_id>/start
/api/calibrations/<ca_id>/stop
/api/calibrations/<ca_id>/status
/api/calibrations/<ca_id>/marker
/api/calibrations/<ca_id>/marker/<x>
/api/calibrations/<ca_id>/marker/<x>/<y>
/api/calibrations/<ca_id>/marker/<x>/<y>/<z>
/api/calibrations/<ca_id>/description
/api/recordings
/api/recordings/<rec_id>
/api/recordings/<rec_id>/start
/api/recordings/<rec_id>/pause
/api/recordings/<rec_id>/stop
/api/recordings/<rec_id>/status
/api/recordings/<rec_id>/segments
/api/recordings/<rec_id>/segments/<seg_id>
/api/recordings/<rec_id>/description
/api/system
/api/system/conf
/api/system/conf/description
/api/system/status
/api/system/eject
/api/system/reboot
/api/system/wlan
/api/system/wlan/reset
/api/system/wlan/scan_networks
/api/system/wlan/description
/api/upgrade
/api/stats
/api/stats/tslv
/api/identify
/api/events
```

Tobii Pro Support

EUROPE / GLOBAL
Phone (SWE): +46 8 522 950 10
Phone (GER): +49 69 24 75 03 4-27
support@tobii.com
Support hours: 8 am - 6 pm
Between July-August: 9 am - 5 pm
(Central European Time, GMT +1)

NORTH AMERICA
Phone: +1 703 738 1320

support.us@tobii.com
Support hours: 8 am - 8 pm
(US Eastern Standard Time, GMT -6)

JAPAN
Phone: +81 3 5793 3316

support.jp@tobii.com
Support hours: 9 am - 5.30 pm
(Japan Standard Time, GMT +9)

CHINA
Phone: +86 180 1558 5168

support.cn@tobii.com