Pro Glasses 3
Developer Guide

Pro Glasses 3 Developer Guide

Version 1.0
10/2020

Please visit Tobii Pro Connect at connect.tobiipro.com for updated versions of this document.

Table of Contents

1 Introduction

This document is intended for readers who want to integrate Tobii Pro Glasses 3 into their own software or research processes, either using the live API to control the glasses hardware, or in the analysis phase, to process data that is captured by the glasses.

Readers who plan to integrate using the Live API are assumed to have software development knowledge and understand concepts like REST API, asynchronous calls, JSON objects, etc.

Tobii Pro Glasses 3 has an API that can be accessed from most modern programming languages. For this reason, this document will be language agnostic and only explain the concepts and workflows needed when developing software with the glasses. There is a reference implementation of most workflows in JavaScript/HTML in the Tobii Pro Glasses 3 example web client.

1.1 Units and definitions

The following units and definitions are used throughout the API:

- **Timestamps** are always in seconds with decimals if nothing else is indicated.
- **Gaze2D** is in normalized video coordinates, (0,0) corresponds to the top left corner of the scene camera video and (1,1) corresponds to the bottom right corner.
- **Gaze3D** is a position in millimeters from the scene camera, X is positive to the left, Y is positive up, and Z is positive forward.
- **Gaze Origin** is a position in the same coordinate system as Gaze3D.
- **Gaze Direction** is a normalized vector that starts in the gaze origin of the respective eye, same coordinate system as Gaze3D.
- **Pupil diameter** is measured in millimeters.
- **Accelerometer** is measured in meter/second^2 (m/s^2).
- **Gyroscope** is measured in degrees/second (°/s).
- **Magnetometer** is measured in microtesla (µT).

1.2 Document conventions

In this document, the network address of Glasses 3 will be written as "<g3-address>". You can replace this with an IPv6 address, an IPv4 address, the hostname (<serialnumber>) or the mDNS address record of your recording unit (<serialnumber>.local).

1.3 Additional resources

This document does not list all operations in the Glasses 3 API.

Pro Glasses 3 has a built in API reference documentation. Point your web browser to the address of the Glasses 3 device (http://<g3-address>). Here you will find an API browser that allows you to execute any of the operations in the API directly on the device. There are also reference implementations in JavaScript/HTMLfor many of the common use cases and scenarios (live view, replay, calibration, etc).

2 API Concepts

Pro Glasses 3 has a REST API that allows you to communicate with the recording unit to start/stop recordings, perform personal calibrations, get live data streams, etc. that can be accessed either through HTTP requests or WebSocket communication.

2.1 WebSocket

The WebSocket connection is based on sending messages with JSON objects across the socket. Since WebSocket offers asynchronous communication (as opposed to request/response in HTTP communication) each request needs to contain a unique id that will be included in the response message that is sent back over the WebSocket.

The address to connect the WebSocket is `ws://<g3-address>/websocket`, and you must specify the sub protocol “g3api” when you connect.

Special note for .net developers: Due to an incorrect HTTP header “`Content-length:0`” from Pro Glasses 3 in one of the hand-shake messages when creating the WebSocket, the .net `ServicePoint` will immediately schedule the WebSocket for cleanup. The default timeout for this cleanup is 100 seconds. Once cleaned, the WebSocket will no longer receive or send any messages. Until this incorrect header has been removed, the easiest workaround, is to change the default idle time to a high value: `ServicePointManager.MaxServicePointIdleTime = int.MaxValue;`

2.2 HTTP

To make an API request over HTTP, you combine the root URL of the API with the path of the API element you want:

```
http://<g3-address>/rest/<path-to-api-element>
```

Pro Glasses 3 does not support secure/encrypted HTTP (HTTPS).

2.3 Objects

All functionality in the API is grouped into objects. The available objects are listed as children of the root object in the API and include:

- `calibrate` - functionality for making user calibrations
- `neighborhood` - functionality for detecting other glasses on the network using Zeroconf
- `network` - configuration for network settings, both LAN and wireless network settings
- `recorder` - functionality for making recordings
- `recordings` - lists the recordings that are available on the SD card and their metadata

- `rudimentary` - simple interface for accessing data streams and scene camera images during development, testing and debugging
- `system` - system information such as serial number, system time/time zone. Child objects for accessing battery and SD-card information
- `upgrade` - functionality related to firmware upgrade
- `webrtc` - live view video and gaze data over WebRTC

For a full overview of all objects and their usage, please refer to the reference implementation and API reference documentation in the Pro Glasses 3 example web client. For more information, read Additional resources.

2.4 Properties

Properties are read only or read/write values in the API. Each property belongs to an object and is accessed with the path of the object followed by a period "." and the property name.

Syntax: `<path to parent object>.<property name>`

Example: `system.recording-unit-serial`

2.4.1 Read a property value

To read a property, use a HTTP GET-request with the full URL of the property. The response will contain the value of the property.

Example HTTP:

HTTP GET	`http://<g3-address>/rest/system.recording-unit-serial`
Response	`"TG03B-080200004381"`

Example WebSocket:

<table>
<tr><td>Send</td><td><pre>{
  "path":"system.recording-unit-serial",
  "id":22,
  "method":"GET"
}</pre></td></tr>
<tr><td>Receive</td><td><pre>{
  "id":22,
  "body":"TG03B-080200004381"
}</pre></td></tr>
</table>

2.4.2 Write a property value

To write a new value to a read/write property, use a HTTP POST request with the new value as the body. The response when setting a property value is a boolean that indicates if the setting of the value was successful or not.

Example writing a property with HTTP request:

HTTP POST	`http://<g3-address>/rest/recorder.visible-name`
Body	`"Rec1"`
Response	`true`

Example writing a property with WebSocket connection:

Send	{
	"path":"recorder.visible-name",
	"id":39,
	"method":"POST",
	"body":"Rec1"
	}

Receive	{
	"id":39,
	"body":true
	}

2.5 Actions

Actions are used to execute commands through the API. Examples are starting and stopping a recording, deleting a recording, creating a WebRTC-session. Just like properties, actions also belong to objects.

Syntax: <path to parent object>!<action>

Example: recorder!start

When calling an action, you must always provide the arguments as the body of your request. The body should be a JSON array. If there are no arguments, the body should be an empty array ([]). Failure to send a valid body will result in an error.

The result of the action is returned as a JSON value/object. For HTTP requests, it will be the entire response, for WebSocket requests, it will be the body property of the matching response object.

Execute an action with HTTP request example:

HTTP POST	http://<g3-address>/rest/recorder!start
Body	[]
Response	true

Execute an action with WebSocket connection example:

Send	{
	"path":"recorder!start",
	"id":17,
	"method":"POST",
	"body":[]
	}

Receive	{
	"id":17,
	"body":true
	}

Execute an action with parameters using a HTTP request example:

HTTP POST	`http://<g3-address>/rest/recorder!meta-lookup`
Body	`["MyKey"]`
Response	`"c2FtcGxlIGNvbnRlbnRzCg=="`

Execute an action with parameters using WebSocket example:

Send	`{ "path":"recorder!meta-lookup", "id":64, "method":"POST", "body":["MyKey"] }`
Receive	`{ "id":64, "body":"c2FtcGxlIGNvbnRlbnRzCg==" }`

`"c2FtcGxlIGNvbnRlbnRzCg=="` is the Base64 encoded representation of the string "`sample contents`"

If an action fails, it can provide additional information about the failure in an error-info property in the response. This is an example of trying to start a recording without an SD-card inserted:

HTTP request:

HTTP POST	`http://<g3-address>/rest/recorder!start`
Body	`[ ]`
Response	`False`
Response headers	`X-g3-action-error = resource;/system/storage`

WebSocket request:

Send	`{ "path":"recorder!start", "id":17, "method":"POST", "body":[] }`

Receive	`{ "id":17, "body":false, "error-info": {"type":"resource","path":"/system/storage"} }`

2.6 Signals

Signals allow for data to be pushed asynchronously from the glasses to the API client. While it is possible to subscribe to signals using HTTP requests, it is not advised since it will only allow a single message per subscription. The risk of losing messages before the client has placed a new subscription is significant. To subscribe to a signal using WebSocket, send a message that contains the path and a unique id. The response will contain the same id and the body will contain the identity that each data package in the signal will contain.

Send	`{ "path":"system/storage:state-changed", "id":28, "method":"POST", "body":null }`
Receive	`{ "id":28, "body":454 }`

This response tells us that future data packages that are pushed as a part of the `storage:state-changed` signal will have the signal id 454. If the SD card is removed, the following messages will be sent by the RU:

Receive	`{ "signal":454, "body":["good", "not-available"] }`
Receive	`{ "signal":454, "body":["unknown", "not-inserted"] }`

The values in the body comes from the API properties `storage!space-state` (with values `"good", "low", "verylow" and "unknown"`) and `storage!card-state` (`"not-inserted", "not-available", "read-only", "available", and "failure"`). If any of these properties changes, the signal will send a notification. Other signals like a gaze-signal will send new values much more frequently.

2.6.1 HTTP vs WebSocket

All functionality in the API can be used over either HTTP requests or WebSocket requests. Using WebSockets have the following advantages:

- Signals can support multiple responses
- Less resource intensive

3 Workflows

3.1 Discovery

Discovery is the process of finding the network address for a Pro Glasses 3 unit once it is connected to the same network as the API client.

3.1.1 Zeroconf

Pro Glasses 3 can be located using Zeroconf. The device exposes the following services:

Service:	**_tobii-g3api._tcp.**
Address	<ip6-address>
Address	<ip4-address>
Port	80
Text properties:	
path	/rest/

REST API clients should use the path property of the Zeroconf service announcement to build the base path of the API. If the path to the REST API changes in future firmware releases, the Zeroconf service announcement will provide the correct value.

Service:	**_rtsp._tcp**
Host	<serialnumber>.local
Address	<ip6-address>
Address	<ip4-address>
Port	8554
Text properties:	
path	/live/all
Recordings	/recordings

3.1.2 Hostname

The device will have the same hostname as its own serial number. This hostname will be exposed to external DNS servers, so if the DNS server on the network that the glasses is connected to accepts hostnames from clients, this name can be used to access the API.

3.1.3 mDNS

Pro Glasses 3 will register itself on any network with a unique hostname based on the serial number, and it will present the mDNS-name `<serialnumber>.local` (i.e. `TG03B-080200004381.local`). This name can be used to access the REST API (`http://TG03B-080200004381.local/rest`) or to set up a WebSocket connection (`ws://TG03B-080200004381.local`).

3.1.4 Wireless access point

The default network setting of Pro Glasses 3 is to start a wireless access point that any client can connect to. When connected to the Pro Glasses 3 access point, the device itself will always have the same IPv4 address (192.168.75.51).

SSID	<serialnumber>
Password	TobiiGlasses
Subnet Mask	255.255.255.0
Default gateway	192.168.75.51
DNS server	192.168.75.51
DHCP server	192.168.75.51
DHCP Lease time	24h
NTP	Not available
Frequency	2.4 GHz

When configured as access point the network will provide a DNS-server with a DNS record for the hostname (serial number).

The bandwidth available through the wireless network is ~20mbit/s. That means that it will not be possible to stream live video to multiple clients without significant risk of packet/frame loss.

3.2 Live view

Tobii Pro Glasses 3 support live view of the scene camera video over either WebRTC or RTSP. Both protocols offer relatively low latency (~0.2s) for the video and access to both scene camera video, eye camera video (4 camera images per video frame) and audio stream. The scene camera video stream has no gaze overlay.

3.2.1 WebRTC

WebRTC is a video streaming protocol designed for use in a browser environment. WebRTC is supported in the current versions of all major browsers (Chrome, Firefox, Safari, Opera, Edge) and is the same protocol that is used in many video conference / video calling solutions.

The WebRTC implementation in Pro Glasses 3 is verified to be compatible with Chrome and Firefox on Windows, Linux and Mac.

WebRTC is designed to handle connection of clients over the internet and through various network topologies with firewalls and NAT routers, Pro Glasses 3 currently only supports setting up a WebRTC session over a local network using the REST API.

Set up WebRTC

The setup sequence looks like this:

Operation:	Client code	Glasses 3 response
Create RTCPeerConnection in the browser	**JS:** `this.peer = new RTCPeerConnection(rtc_configuration);`	
Subscribe to local ICE candidates	**JS:** `this.peer.onicecandidate = (ev) => this.new_local_ice(ev);`	
Subscribe to peer tracks	**JS:** `this.peer.ontrack = (ev) => this._peer_track(ev);`	

Operation:	Client code	Glasses 3 response
Create the WebRTC session on Pro Glasses 3	`API: var uuid = webrtc!create([])`	uuid of WebRTC session
Subscribe to remote ICE candidates	`API: webrtc/<uuid>:new-ice-candidate`	
Request remote offer	`API: var offer = webrtc/<uuid>!setup([])`	SDP offer

Send remote offer to browser	`JS: this.peer.setRemoteDescription (new RTCSessionDescription({`
	 `type:"offer",`
	 `sdp:offer.to_sdp()`
	 `});`	

Send local ice candidates to RU	`API: webrtc/<uuid>!add-ice-candidate ([index, candidate])`	
Add remote ice candidates to browser	`JS: this.peer.addIceCandidate(candidate)`	
Create local answer	`JS: var answer = this.peer.createAnswer();`	
Send the local answer to the RU	`API: webrtc/<uuid>!start([answer])`	
Hook up the peer track to a video element		

Once the live view is active, it is important to let the glasses know you are still listening by calling the `webrtc/<uuid>!keep-alive` action on a regular basis. This call should be executed at least once every 5 seconds. If the keep-alive method has not been called for 20 seconds, the WebRTC session will assume that the client has lost the connection and terminate. In this case, the `webrtc-session:timed-out` signal will send an event that indicates this.

There is a reference implementation of setting up a live WebRTC stream in the example web client.

WebRTC data channels (added in FW 1.11)

When a working WebRTC connection has been established, there will be data channels available for receiving gaze data, events, IMU data, and syncport events.

Additional information about WebRTC

Chrome and other browsers try to avoid exposing local IP-addresses to the JavaScript environment, and will therefor place anonymous mDNS addresses it the ICE candidates:

```
candidate:3178462403 1 udp 2113937151 b1feeef5-febf-4714-b1ee-7a2b35cfdfcc.local 52432 typ host generation 0 ufrag jhdo network-cost 999
```

These will not always be resolved correctly by the WebRTC server on the glasses. There is an API action `!remote.host` that can be used to query the glasses for your own IP-address and then replace any anonymous address in the contents of the ICE candidate with your explicit address. This is exactly what the “Resolve ZeroConf in Browser” checkbox in the example web client Live View tab does.

Browsers do not always include link-local IPv6-addresses in the ICE candidates. On a direct ethernet cable connection it is more robust to use link-local IPv4 addresses when connecting to the glasses.

3.2.2 RTSP

RTSP is a video streaming protocol that doesn’t require as much setup as WebRTC but is primarily suited for native applications rather than web applications. To start the live RTSP stream of the Pro Glasses 3 system, just connect an RTSP client to `rtsp://<g3-address>:8554/live/all`.

The RTSP-stream contains three separate media streams:

- Payload 96: Scene camera video, h264, 25fps, 1920x1080px
- Payload 97: Microphone audio, mpeg3, 24khz, 32 bits/sample, mono
- Payload 98: Eye camera video, h264, 50fps, 1024x256px

Data streams available:

- Payload 99: Gaze data
- Payload 100: Sync in/out
- Payload 101: IMU data
- Payload 102: Events

The media streams have been tested with VLC, an open source video player available for multiple operating systems including Windows, Mac OS X, Linux, Android and iOS. To open a live video stream in VLC, go to the main menu and select ”Media” and then ”Open network stream”.

3.2.3 Render gaze in live view

The scene camera stream does not have any gaze overlay in it, only the images that comes from the scene camera sensor. This allows a client to decide the look and feel of the rendered gaze marker. In both WebRTC and RTSP, there are additional data channels that provide live gaze samples. The samples contain complete eye tracking information (2D gaze, 3D gaze, gaze origin, gaze direction and pupil diameter). The data for 2D gaze will be in relative video coordinates and can be used to render a marker on the video that represents gaze position.

In live view it is often preferable to perform a bit of noise reduction on the gaze position before overlaying it on the scene camera image.

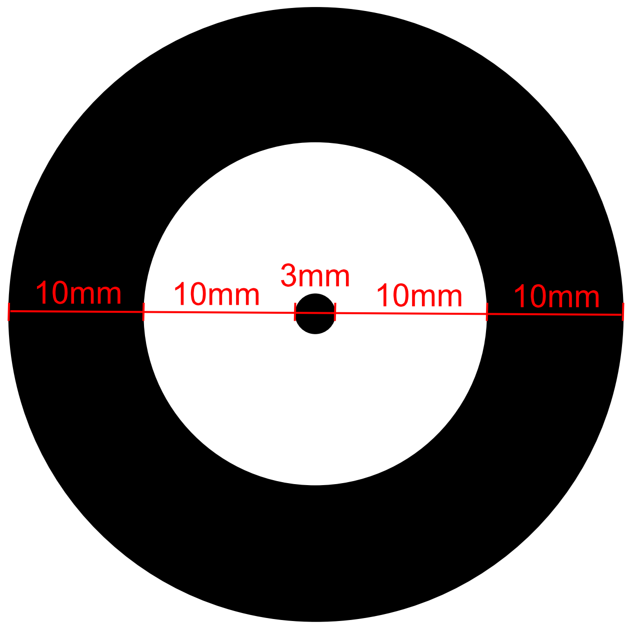
In FW version 1.9 it is not possible to get exact sync of gaze/video in WebRTC. Rendering the latest gaze sample is currently the best option. Sync information will be provided in future FW updates.

The data streams in RTSP have timestamps that allow synchronization with the media streams.

3.3 Calibration

Before eye tracking a person, the person needs to be calibrated. This procedure requires the person to look in a known direction and allowing the eye tracker to compute several properties of the persons eyes. Pro Glasses 3 comes with preprinted calibration markers for this purpose. The marker should be positioned in front of the person at around 50-100 cm and held still before the calibration is performed.

The diameters of the circles in the marker are 43mm/23mm/3mm. If a marker with the wrong dimensions is used, the estimated distance to the marker will be wrong and the eye tracking will not be correct. Exact color is less important, but the contrast between the circles and dot must be high, and the outer circle and central dot must be darker than the middle circle.



During the calibration workflow, it is advisable to enable the marker stream and render the marker on the live video to make it easy to varify that the calibration target is correctly identified. This is done by calling the action `calibrate!emit-markers`. It will enable the detection for 3 seconds or until a calibration is completed. While the marker detection is enabled, the signal `calibrate:marker` will emit values for each video frame that has been processed.

The body of the marker stream will be an array of values that looks like this:

```
[1.869804, [-210.64152, 72.88064, 686.21405], [0.64479, 0.40370]]
```

The first value is a timestamp. The second value is the 3D coordinate of the marker (x, y, z) in millimeters from the scene camera. The third value is the 2D coordinate of the marker in normalized video coordinates.

When the marker is correctly identified in the scene camera stream, and the participant is focusing on the calibration target, call the `calibrate!run` action to perform the calibration. If the calibration is successful, the gaze marker overlay should “snap” to the calibration target in the scene camera video.

Detecting the calibration marker in the scene camera video stream is a rather resource/CPU intensive operation and should not be done during recording of eye tracking data as it can negatively impact the possibility to calculate correct gaze data.

Here is a list of reasons that could cause the firmware to fail to detect the marker or incorrectly detect something else as a marker:

- Too low contrast of the marker
- Too dark or too bright to show the contrasts of the marker in the scene camera image
- Reflections in the marker
- Incorrect proportions of the marker
- Marker too close
- Marker too far away
- Other items in field of view that resembles the calibration marker

There is no requirement to subscribe to gaze data or have an active live view during the calibration, but it will make it easier to verify that all is correctly setup before the calibration is performed. Keep in mind that gaze coordinates will be incorrect until a good calibration has been performed. If Pro Glasses 3 is just started, the gaze will be using the default calibration. Once a calibration has been performed, the calibration will be active until a new calibration is performed and will not be affected by starting or stopping recordings. There is currently no action to reset the calibration.

3.4 Make a recording

The normal workflow when making a recording is:

- Make a successful calibration
- Validate in live view that gaze is correctly positioned on the object that the user is looking at
- Start the recording
- Set the name of the recording and other metadata
- Stop the recording.

Between starting and stopping the recording, all data will be stored on the SD card and there is no requirement to have an active live view or even a network connection with a client. If the network connection between the client and the glasses is lost, the recording will continue until a client reconnects and stops or cancels the recording. If no client connects, the recording will continue until Pro Glasses 3 runs out of battery or space on the SD card.

A controlled power down (pressing the power button for 4 seconds, releasing it when the LEDs change from blinking to solid yellow and then wait until the LEDs turn off) will also terminate the recording. Removing the battery, SD card or performing a forced power down (pressing the power button for 12 seconds) will in most cases render the recording corrupt.

3.4.1 Start, stop, cancel

The `recorder!start` action will start storing data on the SD card (scene camera video, eye tracking data, events, metadata and IMU data). The start action can fail for a number of different reasons:

- No SD card
  - `("type":"resource", path:"/system/storage")`
- Not enough space on the SD card
- SD card is read only
  - `(type:"resource",path:"/system/storage")`
- Not enough battery power
- No head unit connected
  - `(type:"internal",path:"/recorder")`
- Already a recording in progress
  - `(type:"not-permitted",path:"/recorder")`
- There is a folder on the SD card called "recorder"
  - `("type":"internal")`

Once the start action has succeeded, the recording will be in progress. At this point the properties of the recorder-object will provide useful information about the ongoing recording:

- `recorder.created` - the UTC time when the recording was started.
- `recorder.duration` - the number of seconds the recording has been in progress.
- `recorder.folder` - the name of the folder that the recording will be stored in once it is completed. This value can be set to override the default behavior. The character set that can be used is limited to what is allowed for file/folder names.
- `recorder.gaze-samples` - the total number of gaze samples that have been recorded, regardless of if gaze was detected or not.
- `recorder.valid-gaze-samples` - the number of successfully eye tracked samples. This can be used together with the gaze-samples property to indicate if there is a loss of tracking without having to subscribe to the gaze signal.
- `recorder.remaining-time` - the time until the recording will be force-stopped. This value will be calculated based on remaining battery, remaining space on the SD card and file system type of the SD card.
- `recorder.timezone` - the timezone that the glasses were configured with at the start of the recording.
- `recorder.uuid` - the unique identifier that can be used to address the recording after it has been stopped.
- `recorder.visible-name` - this property can be used to give the recording a human readable representation.

During the recording, it is possible to set metadata, folder-name, visible-name and to request the saving of snapshots from the scene camera. The data will be stored in a folder on the SD card called "recorder".

Calling `recorder!stop` will stop the recording and finalize all files stored on the SD card.

The stop operation can fail for the following reasons:

- Invalid folder name
  - Folder-name too long
  - Folder-name contains invalid characters
  - Folder-name is “recorder”

Note, that trying to stop a recording when there is no recording in progress will return true.

After the recording has been stopped, the folder with all data from the recording will be renamed from “recorder” to the value of the `folder-name` property. The recording can be accessed via the API with the UUID that was assigned by the recorder object.

Canceling the recording will remove all data stored during the session. It is semantically equivalent to stopping and deleting the recording, but slightly faster.

3.4.2 Visible name vs folder-name

The `recorder!folder-name` property will be used to create a folder on a FAT32/exFAT file system and must therefore be restricted in length and in which characters are allowed (avoid at least the following characters: `0x00-0x1F 0x7F " * / : < > ? \ |`. The `visible-name` property on the other hand can contain any Unicode character.

Default value for both the `folder-name` and `visible-name` is the UTC time when the recording was started, e.g. `20200710T100603Z`

`Visible-name` is suitable to use as a human readable representation of the recording.

When there is no recording in progress, the `folder-name` and `visible-name` will both be null.

When the recording is started, the `folder-name` will assume its default value (see above), and `visible-name` will still be null. When the recording is stopped, if the `visible-name` is still null, the property will get the same value as the `folder-name` (if the `folder-name` has been changed from the default value, the `visible-name` will get the updated folder name).

Both `folder-name` and `visible-name` can only be set in the `recorder`-object while the recording is in progress. Once the recording is stopped, the `visible-name` is a read/write property in the `recording`-object that is created. The `folder-name` can be changed by using the `move`-action on the `recording`-object.

3.4.3 Metadata in recordings

An API client can store arbitrary metadata in a recording using the metadata actions available in the recorder-object and the recording-objects. The metadata API consists of the following actions:

```
meta-keys(): <array of keys>
meta-insert(key: string, data: <Base64 encoded data>): boolean
meta-lookup(key: string): <Base64 encoded data>
```

The data that is stored using the `meta-insert` action will be stored in a subfolder of the recording folder on the SD card. The file name will be the same as the metadata key (so file name limitations apply to the name of the key). The key can contain period (".") to make the resulting file have a file extension. Using Base64 encoding means that you can store both binary and text data.

The metadata actions can be used to store additional recording information such as information about the participant or project.

When a recording is stopped, you can no longer use the metadata actions in the recorder-object to read/write metadata, but you can do it with an identical set of actions on the recording-object.

3.4.4 Recording events during a recording (FW 1.10+)

During the recording, clients can send client specific events through the API and get them timestamped and stored as a part of the recording.

A custom event consists of a tag and a JSON object. The tag should indicate what the event represents or how it should be interpreted. The tag could for example be the name of the data source, or the type of data it contains.

To send a custom event to the API, use the action `recorder.send-event ([<tag>, <object>])`.

Example of executing an action with parameters using a HTTP request:

HTTP POST	`http://<g3-address>/rest/recorder!send-event`
Body	`[ "gps-location", { "lat":59.400539, "long":18.038402 } ]`
Respons	`true`

Example of sending a customer event over WebSocket

Send	`{ "path":"recorder!send-event", "id":99, "method":"POST", "body":[ "gps-location", { "lat":59.400539, "long":18.038402 } ] }`
Receive	`{ "id":99, "body":true }`

The GPS location can change continuously during the recording, and each time the event is sent to the API, it will be timestamped and stored. It will also be sent to any subscriber of the event-signal (available in WebRTC session objects and the rudimentary object).

The signal would look like this assuming that the signal was assigned signal id 930 when the subscription was created (the values that comes from the send-event action in bold font):

```
{
  "signal":930,
  "body":[
    81.410,
    {
      "tag":"gps-location",
      "object":{
        "lat":59.400539,
        "long":18.038402
      }
    }
  ]
}
```

3.4.5 Take snapshots during the recording

Snapshots are JPEG images taken from the scene camera that are stored on the SD card. By default, a single image is saved in the beginning of the recording. Using the recorder!snapshot action, additional snapshots can be taken during the recording. They will be stored in the recording folder and the timestamp of each snapshot is available in the recording.g3 file. Snapshots are stored with full resolution. No additional EXIF-metadata is stored in the image file.

3.5 Replay

The recording data stored on the SD card can be accessed in several different ways.

3.5.1 Accessing recording data over HTTP

To access the recording data over HTTP, you can get the base path for the recording from the API property recordings/<uuid>.http-path.

Example:

```
http://192.168.75.51/recordings/0b5aaf39-1407-4a2b-825c-a12914ab1783
```

This request will return the contents of the file recording.g3 in JSON format.

A description of the recording.g3 file is available in the appendix, File structure on SD card.

This JSON-object will contain information about the related files such as scene camera video and gaze data file (currently "scenevideo.mp4" and "gazedata.gz"). The file names may change in the future, so it is advised to read the actual file name using the properties in the JSON file.

```
http://<g3-address>/recordings/<uuid>/scenevideo.mp4
http://<g3-address>/recordings/<uuid>/gazedata.gz
```

Adding the URL parameter `"use-content-encoding=true"` will add headers to the response that will make a browser decode the gzip compression on the fly and allow you to treat the file as a text file.

```
http://<g3-address>/recordings/<uuid>/gazedata.gz?use-content-encod-
ing=true
```

The content of the gaze file is a text file that has one JSON object per line (here formatted with line breaks for readability):

```
{
  "type":"gaze",
  "timestamp":681.751,
  "data":{
    "gaze2d":[0.535, 0.477],
    "gaze3d":[-32.499, -4.491, 345.359],
    "eyeleft":{
      "gazeorigin":[30.221, -11.798, -23.935],
      "gazedirection":[-0.160, 0.0432, 0.986],
      "pupildiameter":2.653
    },
    "eyeright":{
      "gazeorigin":[-31.116, -12.392, -21.406],
      "gazedirection":[-0.011, -0.00270, 0.9999],
      "pupildiameter":2.374
    }
  }
}
```

The timestamps of gaze data, events and mems data will correspond to the timestamps of the video.

In the Example web client, there is a reference implementation of showing recording information and replaying recordings with gaze overlay using the HTTP method in the tab called "Storage".

3.5.2 Replaying a recording with RTSP

The RTSP server on Pro Glasses 3 allows replaying a recording that is on the SD card.

```
rtsp://<g3-address>:8554/recordings?uuid=<uuid>
```

Replay via RTSP allows seeking in the video using the standard RTSP API.

3.5.3 Replaying a recording with WebRTC

WebRTC can be used to replay existing recordings as well. Create a WebRTC session using the action `webrtc!play(<uuid>)` and provide the `uuid` of the recording as a parameter for the action. For details on how to setup the WebRTC session, see the section about live view.

Replay via WebRTC does not currently support seeking in the video.

In the Example web client, there is a reference implementation of showing recording information and replaying recordings with gaze overlay using the WebRTC method in the tab called "Recordings".

3.6 Time synchronization and NTP

When NTP (Network Time Protocol) is enabled through the API (`system.ntp-is-enabled`, default on) Pro Glasses 3 will regularly try to find a NTP-server if it is connected to an external network over LAN or Wi-Fi. If the RU is connected at boot time, it will usually sync immediately. If it has been running without a network, it can take significant time until the NTP-sync happens (30-60 minutes). Use `system.ntp-is-synchronized` to determine if the glasses are synchronized or not. If the RU is synced to an NTP-server and is subsequently disconnected form the network where the NTP-server is available, it will still consider itself synced for a time (8+hours). After reboot, the RU will no longer consider itself synced but the clock will still be relatively correct (the glasses system clock has a separate battery backup that lasts for several weeks).

To set the device clock through the API, the client must ensure that the NTP-functionality is disabled (regardless if the RU is synced or not), otherwise the attempt to set the time will fail. To do this, you set `system.ntp-is-enabled` to false. Setting the time zone will always be possible, regardless if the RU is NTP-synced or not.

When setting the system time, an explicit time zone needs to be specified. The format for setting the time to a UTC time looks like this:

- `yyyy-mm-ddThh:mm:ss[.dddddd]Z`
- example: `2020-07-16T07:56:04.089725Z`

Note that leading zeros are required. Z is the abbreviation of UTC time zone (aka the “Zulu time zone”). Decimal seconds are optional.

To set the system time using local time, an explicit “time zone designator” should be provided and should be specified as an offset from UTC. The equivalent of the above time with a time zone for Sweden/DST (2h ahead of UTC) would be:

- `2020-07-16T09:56:04.089725+02:00`

In most cases, it is easier to convert the local time to UTC before setting the time.

3.7 Time zone

Setting the time zone will have the following effects:

- `recorder.timezone` and `recordings/<uuid>.timezone` properties will indicate the time zone that was active when the recording was started
- Creation date/modified date of files on the SD card will be in the configured time zone

Setting the time zone does NOT affect:

- The default folder names of recordings
- The value of the `recorder.created`-property
- The value of the `recordings/<uuid>.created`-property
- The `system.time`-property

All of the above will always use UTC notation, regardless of the time zone setting.

4 SD card file system

Pro Glasses 3 supports SD cards formatted with either FAT32 or exFAT. The FAT32 file system is restricted to files smaller than 4GB (~1h 40min of G3 recording). exFat supports much larger files and is recommended for long recordings with an external power supply to the Pro Glasses 3 recording unit.

5 Streams and timestamps

Timestamp in all data streams in the Glasses 3 API originate from the same physical clock, meaning that they will never drift relative to each other, but streams originating from different API objects will have different offset to the internal clock. This means that a gaze sample in the gaze signal of a `webrtc` object will not have the same timestamp as the same sample in the gaze signal of the `rudimentary` object.

In the files in the recording folder on the SD card, all streams will begin from 0 (zero represents the first frame of the scene camera video).

Each data stream will always have strictly increasing timestamp.

Each API object will use an offset that is calculated at the creation of the API object (this will be time 0).

6 Network configuration

Glasses 3 is equipped with both an ethernet network port and a wireless adapter. The mechanism for configuring network settings is to create a network configuration object, set the properties of the new configuration, save it and finally activate it as the current configuration.

The example web client has a convenient web page that simplifies manual creation of network configurations and shows the current status of the ethernet and wireless network adapters.

6.1 Ethernet

For the ethernet adapter, there are two predefined network configurations:

`default`	Configuration for networks with a DHCP server. Both IP4 and IP6 will try to request a DHCP address. If there is non DHCP-server, the network adapter will be regularly restarted to get contact with a DHCP server.
`default-link-local`	Configuration for direct ethernet connection between the glasses and a client. Both IP4 and IP6 interfaces will be configured to assign link-local addresses (`169.254.x.y / fe80::xxxx:yyyy:zzzz:vvvv`).

If you need a custom configuration you can create a new configuration and set all the network parameters through the API.

6.2 Wireless

To connect to an external wireless network, create a new network configuration using the `network/wifi!create-config` action. The return value of this action will be the UUID of the new network configuration. For most networks it is sufficient to configure the following properties:

- `network/wifi/configurations/<uuid>.ssid-name` = ssid of your wireless network
- `network/wifi/configurations/<uuid>.security` = wpa-psk
- `network/wifi/configurations/<uuid>.psk` = password of your wireless network

If the network is an open network, only the `ssid-name` property needs to be configured, the rest can be left with default values.

The `ssid-name` property accepts most commonly used SSIDs as a utf-8 encoded string. However, the SSID of a network is really a byte array and not a string and cannot always be represented as a string. The property `network/wifi/<uuid>.ssid` accepts a Base64 encoded version of the byte array that represents the SSID.

For more advanced networks or networks without DHCP capabilities, you might need to set additional properties for name servers, fixed IP-addresses etc.

When the required properties have been configured, the configuration should be saved using `network/wifi/configurations/<uuid>!save`, Finally the new configuration can be activated using `network/wifi!connect` with the UUID of the network configuration as the only parameter (Body = `["<uuid>"]`).

To reset the network settings to the factory defaults, call `network!reset`, this will remove all manually added network configurations for both `ethernet` and `wifi`. After resetting, the ethernet interface will support DHCP networks, and the wireless interface will provide an access point.

Example workflow to connect Glasses 3 to a wireless network:

```
var uuid = network/wifi!create-config(["MyNetworkConfig"])
network/wifi/configurations/<uuid>.ssid <= "my-ssid"
network/wifi/configurations/<uuid>.security <= "wpa-psk"
network/wifi/configurations/<uuid>.psk <= "network password"
network/wifi/configurations/<uuid>!save
network/wifi!connect(["<uuid>"])
```

Alternative method by using the functionality to scan for networks:

`network/wifi!disconnect` // required to scan for available networks

`network/wifi!scan`

select an appropriate network definition from the childnodes of

`network/wifi/networks`

```
network/wifi!connect-network(
     ["<uuid of selected network>", "password"])
```

If there is no configuration with a matching SSID, a new configuration will be created and the SSID and password will be saved with the configuration for future use. If there is already a configuration, the password will be reused if you do not provide a new password. A password override will not be saved to an existing configuration. To connect to an open network, there is no need to provide a password.

The wireless interface will operate on 2.4 GHz when acting as an access point, but when connecting to external networks, it supports both 2.4 GHz and 5 GHz networks.

7 Migrate from Glasses 2

7.1 Projects/participants/calibrations

Glasses 3 does not have the concept of projects or participants, but if your client has these concepts the information can be stored as custom metadata in the recordings using the metadata interfaces in the recorder object. There is no support in Glasses 3 to store participant/projectdata on the SD card except as a part of a recording.

7.2 Live view

Glasses 2 supported two different protocols for live viewing. The first protocol that was supported was MPEG-TS over UDP with gaze data streaming over a persistent HTTP stream. In more recent firmware releases support was added for RTSP with gaze transmitted as a data channel in the RTSP stream.

Glasses 3 supports RTSP with gaze transmitted in data channels, but the structure of the gaze packages has changed to align with the rest of the Glasses 3 API. In addition to RTSP, Glasses 3 also supports streaming video and data over WebRTC.

7.3 Replay

Glasses 2 provided access to the recording files using a file share with guest access (no password). In Glasses 3, all files belonging to a recording are provided over HTTP.

7.4 Discovery

Glasses 2 initially came with a custom discovery mechanism implemented by broadcasting custom UDP packages in a dedicated port. Later versions of the firmware also added support for discovery via Zeroconf. Glases 3 only supports discovery via Zeroconf, using a separate service type from Glasses 2 to be able to easily identify if a device is Glasses 2 or Glasses 3.

7.5 REST API

The REST API in Glasses 3 is syntactically very different from Glasses 2 even if they are semantically similar. We have tried to keep the number of concepts down to reduce the complexity in the Glasses 3 API and stick to standards as much as possible. There is no easy way to convert existing code to from Glasses 2 to Glasses 3.

7.6 Missing features

There are some features in Glasses 2 that have not been enabled/implemented in Glasses 3 yet.

- Gaze based exposure of scene camera
- Saving of eye camera images/video

These features are in the backlog and will be enabled in Glasses 3 as soon as time permits.

8 Firmware upgrade

The recommended way to upgrade firmware on Tobii Pro Glasses 3 is to use Tobii Pro Glasses 3 Controller. This will always give you the latest available firmware. Controller will download and cache the latest firmware whenever it has access to internet and allow firmware upgrade also in off-line mode. If your workflow requires that you update the firmware with a fully automated process, or to a version other than the latest available, it is possible to do that via the upgrade API. Please contact Tobii Pro if you need access to firmware upgrade files. Tobii Pro reserves the right to retract previously released versions of the Glasses 3 firmware without prior notice.

The following should be ensured before performing a firmware upgrade:

- The head unit should be connected to the recording unit
- Sufficient battery or external power connected

Manually, the upgrade can be performed using the following unlisted web page from the example web client: `http://<g3-address>/upgrade.html`

This page also acts as a reference implementation for how to use the RestAPI to send a firmware upgrade file (*.tg3) and monitor the progress. To initiate a firmware upgrade, send the following request

- POST `http://<g3-address>/upgradefile`
- Body: the contents of the tg3-file

During the firmware upgrade, the `upgrade:progress` signal will be sent continuously during the process. The signal contains two numeric values in the range 0..1 that indicates upload and upgrade progress respectively. Upload progress will be sent frequently. Upgrade progress will be sent between each individual upgrade package that is processed on the device.

After a firmware upgrade, the `upgrade:completed` orignal will indicate if the upgrade was successful or not and after that the firmware will be restarted including the wireless access point.

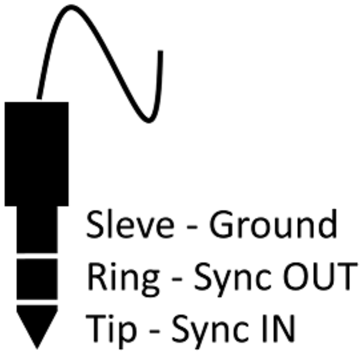
Note that some firmware upgrades include an updated firmware for the head unit. These will take significantly longer to complete and there will be long periods of time without any new data pushed on the `upgrade:progress` signal. Flashing the head unit FPGA firmware typically takes several minutes.

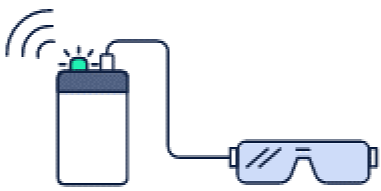
If a head unit with a mismatching FPGA firmware is connected to a recording unit, the firmware in the head unit will be updated immediately and the system will be unresponsive for several minutes.

The firmware upgrade process does not support downgrading the firmware to an earlier release. If this is needed, please contact Tobii Pro Support for assistance.

NB! Please be aware that it is important that the firmware upgrade process is not interrupted. Make sure you have sufficient battery (and possibly an external power source) before initiating a firmware upgrade. If the firmware upgrade process is interrupted, it can leave the device in a bricked state that will require you to send it to Tobii for service.

9 TTL and sync port





Setup description

The Glasses RU connector must be 3-pin 3.5 mm male jack port connector with the following connections:

- Tip: Sync in - Receive events
- Ring: Sync out - Send out events
- Sleeve: Ground

As an example, if we want to receive TTL events in the Glasses: the sleeve of the Glasses port has to be connected to the ground pin of the external device port and the tip has to be connected to the pin of the external device port used to send the TTL events.

Sync out - Send events from Glasses 3

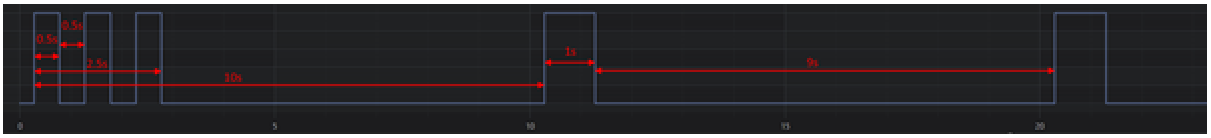
The Glasses 3 always send periodic 1-bit LVTTL signals when a recording starts.

Events: The low state (0) corresponds to 0V and the high state (1) to 3V. Any change from low to high and vice versa will be timestamped in the event data stream:

```
{
  "type":"syncport",
  "timestamp":0.28622,
  "data":{"direction":"out","value":1}
}
{
```

```
  "type":"syncport",
  "timestamp":0.78622,
  "data":{"direction":"out","value":0}
}
```

Sequence: When the recording starts, a sequence of 1/0 is sent three times. Each state lasts 500ms. Starting 10 seconds after the first signal, a 1-second pulse is sent every 10 seconds.



Sync in – Receive events in Glasses 2

The Glasses 3 will timestamp any 1-bit TTL event sent to the Glasses sync-in port.

Events: Any change from low (0: 0V) to high (1: from 2.5V to 3.3V) and vice versa detected in the sync-in port will be timestamped in the event stream:

```
{
  "type":"syncport",
  "timestamp":2.71828182,
  "data":{"direction":"in","value":1}
}
{
  "type":"syncport",
  "timestamp":3.14159265,
  "data":{"direction":"in","value":0}
}
```

Note that the input voltage should not be higher than 3.3V.

Appendix A File structure on SD card

Every recording is stored in a separate folder in the root of the SD card. The folder name defaults to the UTC time of the start recording operation but can be changed while the recording is in progress.

Rearranging or modifying the files in the recording folder could lead to problems replaying the recording with Glasses 3, it could also lead to problems when importing the recording into Tobii Pro Lab for later analysis.

A1 Example `recording.g3` file

```
{
  "uuid":"03e9dfbf-83b8-45dd-8ee7-6fa28b0ba760",
  "name":"My First Recording",
  "version":1,
  "duration":12.767,
  "created":"2020-03-17T11:41:56.852577Z",
  "timezone":"Europe/Stockholm",
  "meta-folder":"meta",
  "scenecamera":{
    "file":"scenevideo.mp4",
    "camera-calibration", {
      "position":[0.0, 0.0, 0.0],
      "rotation":[
        [-0.999,0.014,-0.0207],
        [-0.0139,-0.999,-0.0131],
        [-0.0209,-0.0128,0.999]
      ],
      "focal-length":[917.21,916.74],
      "skew":0.0,
      "principal-point":[961.65,514.30],
      "radial-distortion":[-0.0511,0.0758,-0.0480],
      "tangential-distortion":[-0.000331,0.00000810]
    },
    "snapshots":[
      {
        "file":"snap0.jpg",
        "time":0.970396335
      }
    ]
  },
  "eyecameras":{
    "file":"eyevideo.mp4",
  },
  "gaze":{
```

```
    "file":"gazedata.gz",
    "samples":641,
    "valid-samples":623
  },
  "events":{
    "file":"eventdata.gz"
  }
}
```

A2 Recordings folder

In each recording folder there is one main "`recording.g3`" information file which describes the recording and its current state.

The file uses utf-8 encoding of non ascii characters. The file has no byte order mark (BOM).

It consists of a JSON object with the following structure:

Name	Type	Description
`uuid`	string	A unique UUID representing the recording
`name`	string	The descriptive name of the recording (can be changed through the API using the “visible-name” property)
`version`	integer	The version of the recording.g3 file. Version will be increased when breaking changes occur (element name or element type changes). New elements will not cause version update
`duration`	double	The length of the recording in seconds
`created`	string	UTC date and time of recording start in ISO8601 format
`timezone`	string	The configured time zone at recording start
`sceenecamera`	JSON object	Scene Camera Object (see below)
`eyecameras`	JSON object	Eye Camera Object (see below)
`gaze`	JSON object	Gaze Object (see below)
`imu`	JSON object	IMU Object (see below)
`events`	JSON object	Events Object (see below)
`meta-folder`	string	The name of the metadata folder

A2.1 Scene Camera Object

Name	Type	Description
`file`	string	The name of the scene camera video file or **null** if no video was available
`camera-calibration`	JSON object	A Camera Calibration Object for the scene camera (see below)
`snapshots`	array	Array of Snapshot Objects (see below)

The scene camera file is an MP4 file with the following properties:

Stream 0 - video	
Codec	h.264 - MPEG-4 AVC
Video resolution	1920x1080
Frame rate	25 fps

Stream 1 - audio	
Codec	MPEG Audio layer 1 (mp3)
Channels	Mono
Sample rate	24000 Hz
Bits per sample	32
Bitrate	96 kb/s

A2.2 Eye Camera Object

Name	Type	Description
`file`	string	The name of the eye camera video file or null if no video was available

A2.3 Gaze Object

Name	Type	Description
`file`	string	The file name of the gaze data file
`samples`	integer	Total number of gaze samples in the gaze data file
`valid-samples`	integer	Total number of valid gaze samples in the gaze data file. A gaze sample is considered valid if at least one eye was successfully tracked

A2.4 Camera Calibration Object

Name	Type	Description
`position`	double array	3D Position of the camera
`rotation`	double matrix	The 3x3 rotation matrix
`focal-lengt`	double array	2D Focal Length
`skew`	double	Camera skew
`principal-point`	double array	2D Principal Point
`radial-disortion`	double array	3D Radial Distortion
`tangential-disortion`	double array	2D Tangential Distortion
`resolution`	int array	Pixel resolution

A2.5 Snapshot Object

Name	Type	Description
`file`	string	The file name of the snapshot
`timestamp`	double	Seconds in the recording the snapshot was taken

A2.6 Events Object

Name	Type	Description
`file`	string	The file name of the events data file

A2.7 IMU Obejct

Name	Type	Description
`file`	string	The file name of the IMU data file

A2.8 Meta folder

The folder "meta" contains additional metadata that can be added through the recorder-object in the API.

The following files will always be stored in the metadata folder when the recording is started:

Name	Content	Description
`RuVersion`	string	The Glasses 3 firmware version that was used when the recording was created
`RuSerial`	string	The Glasses 3 serial number of the unit that created the recording
`HuSerial`	string	The serial number of the head unit that was used when the recording was created

Additional metadata stored by the client during the recording will be stored as files with the specified meta data key as the file name.

A3 On disk data format

Data files in the recording folder (gaze, events, imu) all share the same basic structure.

- Text files with LF as the line separator
- One JSON object per line
- Compressed with gzip
- File name ends with .gz

The JSON objects in each file also shares the following structure:

File name	Type	Description
`type`	string	The type of the JSON object
`timestamp`	double	Number of seconds into the recording that the data was sampled
`data`	JSON object	Specific content for each type of sample

A4 Gaze data

If an eye is invalid, the eye object is empty. If both eyes are invalid, the data object is empty.

Field name	Type	Description
`gaze2d`	double array	Horizontal (x) and vertical (y) position of the gaze sample in normalized video coordinates (0-1). 0,0 corresponds to top left corner. 1,1, corresponds to bottom right corner.
`gaze3d`	double array	Position of the vergence point of the left and right gaze vector relative to the scene camera. Measured in millimeters.
`eyeleft.gazeorigin eyeright.gazeorigin`	double array	Position of the eye relative to the scene camera. IPD (inter pupil distance) can be calculated as the distance between the left and right gaze origin

Field name	Type	Description
`eyeleft.gazedirection` `eyeright.gazedirection`	double array	The estimated gaze direction in 3D. The origin of this vector is the gazeorigin for the respective eye.
`eyeleft.pupildiameter` `eyeright.pupildiameter`	double	Diameter in millimeter of the pupil.

Example:

<table>
<tr><td>Gaze sample with valid gaze for both eyes:</td><td><pre><code>{
  "type":"gaze",
  "timestamp":10.21,
  "data":{
    "gaze2d":[0.468,0.483],
    "gaze3d":[37.543,-18.034,821.265],
    "eyeleft":{
      "gazeorigin":[28.799,-7.165,-23.945],
      "gazedirection":[0.0463,-0.0337,0.998],
      "pupildiameter":2.633},
    "eyeright":{
      "gazeorigin":[-28.367,-5.353,-21.426],
      "gazedirection":[0.0436,0.00611,0.999],
      "pupildiameter":2.782}
  }
}</code></pre></td></tr>
<tr><td>Gaze sample with invalid gaze for left eye:</td><td><pre><code>...
  "data":{
    "gaze2d":[0.468,0.483],
    "gaze3d":[37.543,-18.034,821.265],
    "eyeleft":{},
    "eyeright":{
      "gazeorigin":[-28.367,-5.353,-21.426],
      "gazedirection":[0.0436,0.00611,0.999],
      "pupildiameter":2.782}
  }
...</code></pre></td></tr>
<tr><td>Gaze sample with invalid gaze for right eye:</td><td><pre><code>...
  "data":{
    "gaze2d":[0.468,0.483],
    "gaze3d":[37.543,-18.034,821.265],
    "eyeleft":{
      "gazeorigin":[28.799,-7.165,-23.945],
      "gazedirection":[0.0463,-0.0337,0.998],
      "pupildiameter":2.633},
    "eyeright":{}
  }
...</code></pre></td></tr>
</table>

Gaze sample with invalid gaze for both eyes:	… `"data":{}` …

A4.1 Sync Port Event Data

File name	Type	Description
`direction`	string	If change was on "in" or "out" port
`value`	int	The new value, 1 or 0.

Example:

<table>
<tr><td>Sync port sample when sync in changed to 1/high</td><td><pre>{
  "type":"syncport",
  "timestamp":1.13,
  "data":{
    "direction":"in",
    "value":1
  }
}</pre></td></tr>
<tr><td>Sync port sample when sync out changed to 0/low</td><td><pre>{
  "type":"syncport",
  "timestamp":1.13,
  "data":{
    "direction":"out",
    "value":0
  }
}</pre></td></tr>
</table>

A4.2 Event Data

Field name	Type	Description
`tag`	string	A client specific tag name for the data structure format
`object`	JSON object	Client controlled data

<table>
<tr><td>Event example from a hypothetical simulator</td><td><pre>{
  "type":"event",
  "timestamp":4.13,
  "data":{
    "tag":"simulator telemetry",
    "object":{
      "joystick direction":"left",
      "speed":43.2,
      "level":3
    }
  }
}</pre></td></tr>
</table>

A4.3 IMU Data

IMU data is taken directly from one or more IMU sensors. These sensors may work in different frequencies so each IMU sample may contain values from one or more individual sensors.

Field name	Type	Description
`accelerometer`	double array	Acceleration in three dimensions in the coordinate system of the head unit. The values are in m/s2. When the head unit is still, the accelerometer should indicate ~9.8 downward along the z-axis (obey gravity, it's the law)
`magnetometer`	double array	Measurement of the magnetic field near the head unit. This can be used to determine the orientation of the head unit. The values are in micro tesla.
`gyroscope`	double array	The gyroscope measures the rotation of the head unit in three dimensions. The Values are in degrees/second.

Example:

IMU sample with only data from only one sensor	`{ "type":"imu", "timestamp":1.23, "data":{ "magnetometer":[-0.0418,0.229,-0.196] } }`
Sync port sample	`{ "type":"imu", "timestamp":1.33, "data":{ "accelerometer":[-0.0427,-0.920,0.472], "gyroscope":[2.601,0.0822,-0.179] } }`

Pro Glasses 3 Developer Guide v1.0

Support for Your Tobii Pro Device

Get Help Online

Visit Tobii Pro Connect for help with your Tobii Pro device. It contains the latest information about contacting Support, links to our Learning Center, and much more.

Visit connect.tobiipro.com

Contact Your Solution Consultant or Reseller

For questions or problems with your product, contact your Tobii Pro sales representative or authorized reseller for assistance. They are most familiar with your personal setup and can best help you with tips and product training.

Visit tobiipro.com/contact