# Presentation Extension for Tobii Pro

## User's Manual

Version 3.0.1

# Table of Contents

# 1 Introduction

Presentation Extension for Tobii Pro offers the possibility to use Tobii Pro Eye Trackers within Presentation experiments. The extension supports gaze and pupil data as well as automatic and custom calibration.

This manual describes the installation procedure and how to use the Presentation Extension for Tobii Pro. Please review the manual completely and thoroughly before beginning the system Installation.

**Important Note:**

Presentation is not designed as a data acquisition system. Eye data received by Presentation is not saved to disk and does not appear in the logfile. Data should only be transmitted to Presentation if the stimulus display or scenario behavior is dependent on that data.

## 1.1 Supported Eye Trackers

Presentation Extension for Tobii Pro uses Tobii Pro SDK Version 1.4.0 (2018-04-11) and has been tested to work with the following Tobii Pro eye trackers:

- Tobii Pro Spectrum
- Tobii Pro X3-120 (with and without EPU)
- Tobii Pro X2-60

The following eye trackers from Tobii Pro that are compatible with the Tobii Pro SDK should also work:

- Tobii Pro TX300 (Firmware version 1.0.0 or higher)
- Tobii Pro T60XL (Firmware version 2.0.0 or higher)
- Tobii Pro X2-30
- Tobii X60/X120 (Firmware version 2.0.0 or higher)
- Tobii T60/T120 (Firmware version 2.0.0 or higher)

# 2 Installation of the Extension

The objective of this chapter is to describe how to install Presentation Extension for Tobii Pro on your Windows computer.

## 2.1 Overview

Presentation Extension for Tobii Pro is a software extension to Presentation by Neurobehavioral Systems, Inc that allows communication between a Tobii Pro Eye Tracker and Presentation during the run of an experiment. It will allow you to create Presentation experiments or update existing Presentation experiments to function with the Tobii Pro Eye Trackers. The extension also includes a set of sample experiments that can be run directly, or used as a starting point from which to create new experiments.

As Presentation Extension for Tobii Pro uses the eye tracker interface version 2, it is compatible with Presentation Version 11.2 Build 09.24.07 or newer.

## 2.2 Software Installation

Before continuing, be sure that you have administrative rights to install this software on the computer. If you do not have administrative rights, you will be unable to install Presentation Extension for Tobii Pro. If you are unsure of your administrative privileges, contact your System Administrator.

**Install Presentation**

1. Download Presentation from the Neurobehavioral Systems, Inc website (http://www.neurobs.com/menu_presentation/menu_download/current) and save it to your hard disk.
2. Locate and run the installation file on your computer.
3. Follow the prompts in the installation program to provide any required information (e.g. User Name, Company, etc). Usage of Presentation requires a valid Presentation License. See the Neurobehavioral Systems, Inc website for details (http://www.neurobs.com).

**Download Presentation Extension for Tobii Pro**

1. Download the Presentation Extension for Tobii Pro ZIP-file from http://www.vision-space.com and save it to your hard disk.
2. Locate and unzip the content of the file to a location on your computer.

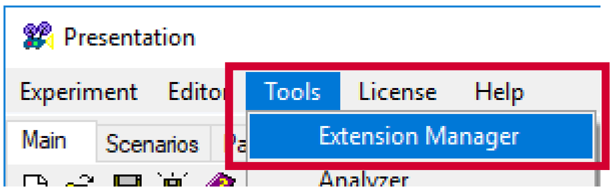
In addition to the extension DLLs this documentation and several sample experiments that show how to use the extension within your own Presentation experiments are copied to the chosen destination.

**Register Presentation Extension for Tobii Pro with Presentation**

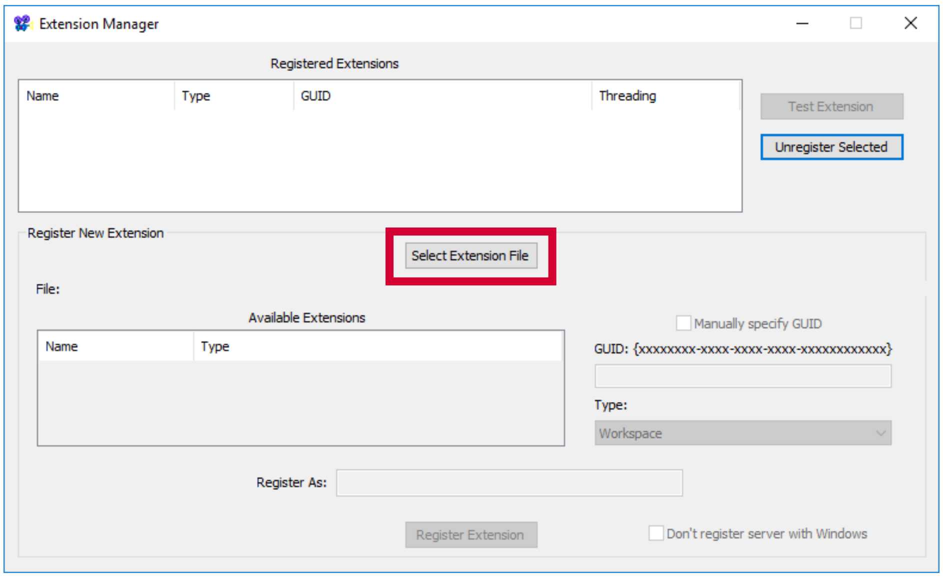
Follow the steps described below to register the extension with Presentation.

1. Launch **Presentation**

2. Select **Extension Manager** from the **Tools** menu



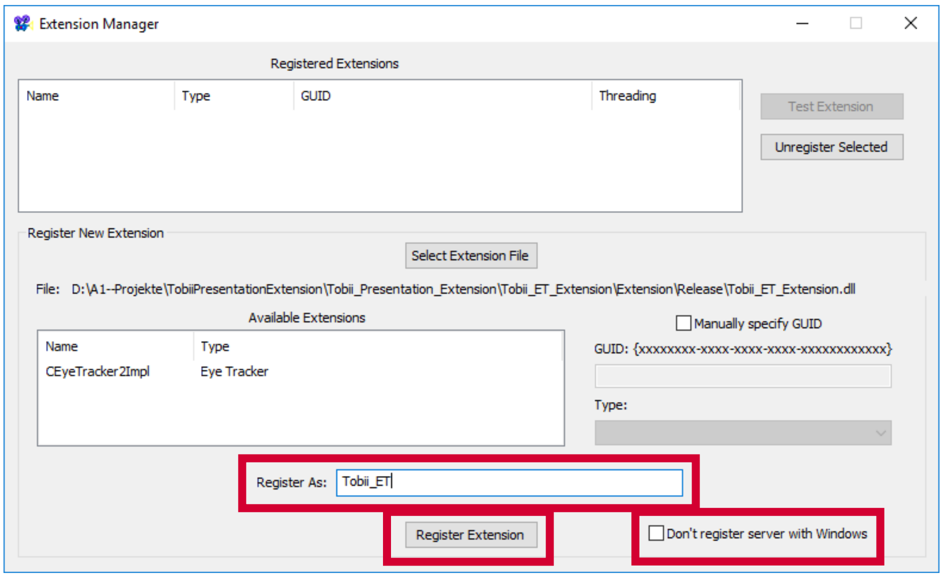
3. **Click** the **Select Extension File** button. A file selection dialog will pop up. Go to the location you chose as destination for the Presentation Extension for Tobii Pro and select the file **TobiiPro_ET_Extension.dll**.



4. Choose a valid unique name (e.g. **Tobii_ET**) for the extension.

   **Do not** select the **Don't register server with Windows** option.

   Click **Register Extension**.



Extension Manager

Registered Extensions

| Name | Type | GUID | Threading |
|---|---|---|---|

Test Extension
Unregister Selected

Register New Extension

Select Extension File

File: D:\A1--Projekte\TobiiPresentationExtension\Tobii_Presentation_Extension\Tobii_ET_Extension\Extension\Release\Tobii_ET_Extension.dll

Available Extensions

| Name | Type |
|---|---|
| CEyeTracker2Impl | Eye Tracker |

☐ Manually specify GUID

GUID: {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}

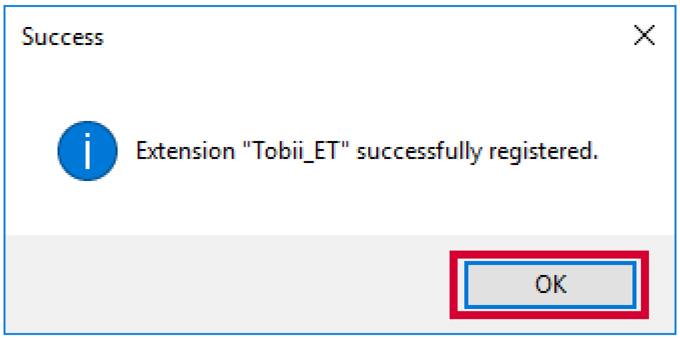
Type:

Register As: Tobii_ET

Register Extension

☐ Don't register server with Windows

5. In case of no error you should see the **Success** message dialog. **Click OK** to close the dialog.

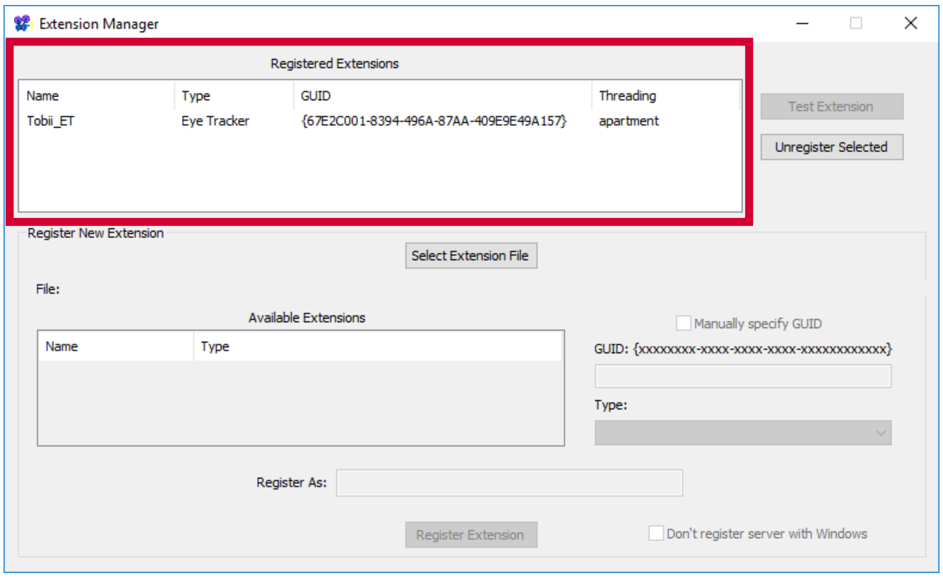


Success

Extension "Tobii_ET" successfully registered.

OK

The extension will be shown in the list of **Registered Extensions.**



Extension Manager

Registered Extensions

| Name | Type | GUID | Threading |
|---|---|---|---|
| Tobii_ET | Eye Tracker | {67E2C001-8394-496A-87AA-409E9E49A157} | apartment |

Test Extension
Unregister Selected

Register New Extension

Select Extension File

File:

Available Extensions

| Name | Type |
|---|---|

☐ Manually specify GUID

GUID: {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}

Type:

Register As:

Register Extension

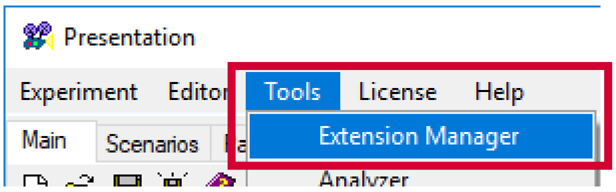
☐ Don't register server with Windows

## Test Presentation Extension for Tobii Pro Version 2.0 with Presentation

Follow the steps described below to test the extension with Presentation.

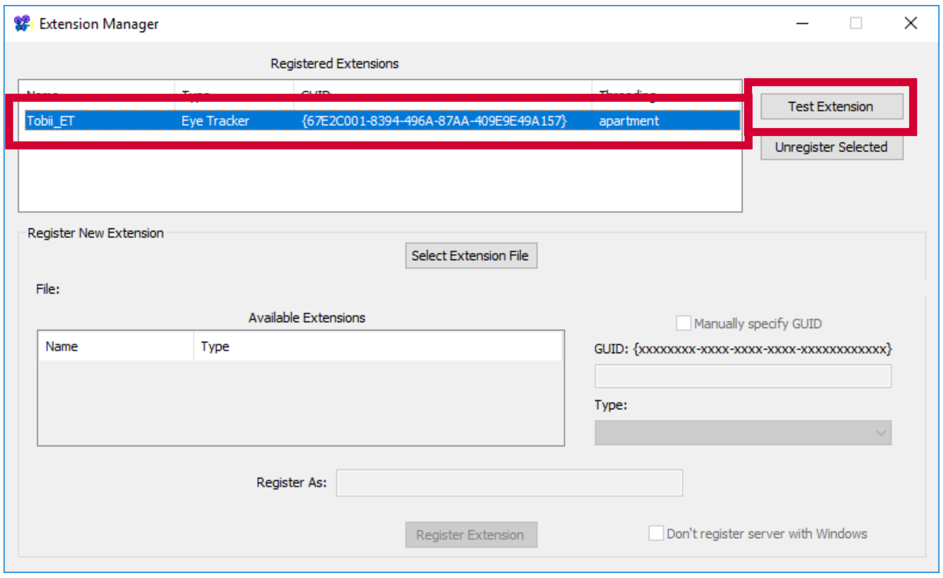
1. Launch **Presentation**

2. Select **Extension Manager** from the **Tools** menu



3. **Select** the **extension** by clicking on it in the list of **Registered Extensions**.

   **Click Test Extension** to open the **Eye Tracker Extension Test Window**.



Extension Manager

Registered Extensions

| | | | |
|---|---|---|---|
| Tobii_ET | Eye Tracker | {67E2C001-8394-496A-87AA-409E9E49A157} | apartment |

Test Extension

Unregister Selected

Register New Extension

Select Extension File

File:

Available Extensions

| Name | Type |
|---|---|
| | |

Manually specify GUID

GUID: {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}

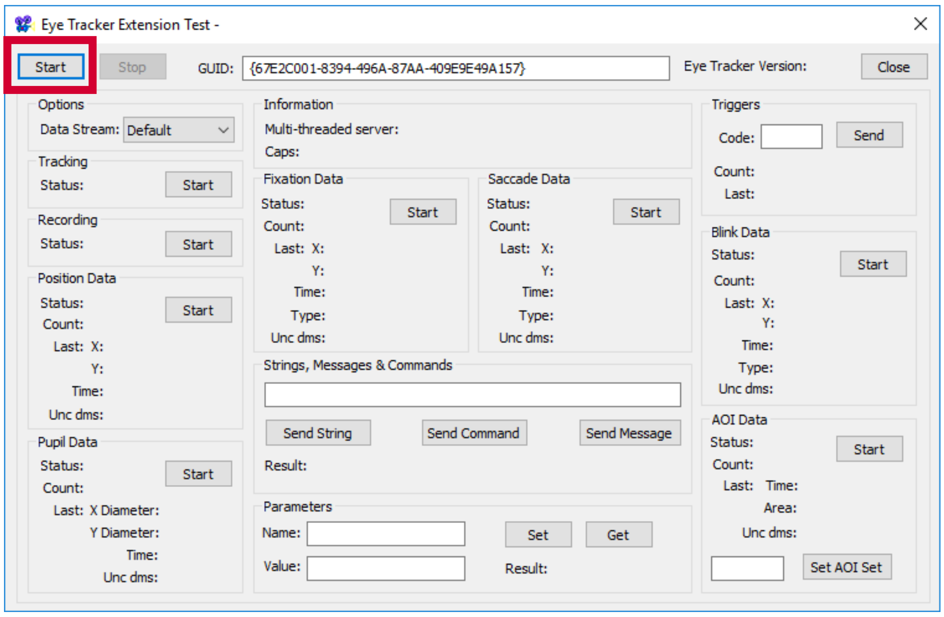
Type:

Register As:

Register Extension

Don't register server with Windows

4. **Click** on the **Start** button to start the extension. Make sure you have at least one supported Tobii Pro eye tracker connected, otherwise you will get an error.



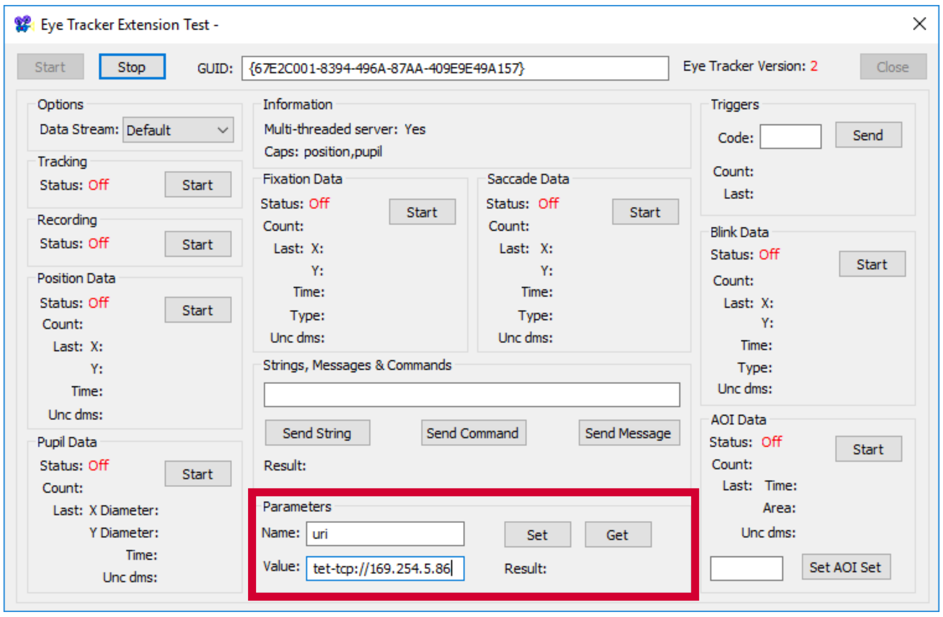
GUID: {67E2C001-8394-496A-87AA-409E9E49A157}

Eye Tracker Version:

Data Stream: Default

5. Optionally set the **Uri parameter** for connecting to the eye tracker.

   See chapter 4.4 for list of supported parameters.



GUID: {67E2C001-8394-496A-87AA-409E9E49A157}

Eye Tracker Version: 2

Data Stream: Default

Multi-threaded server: Yes

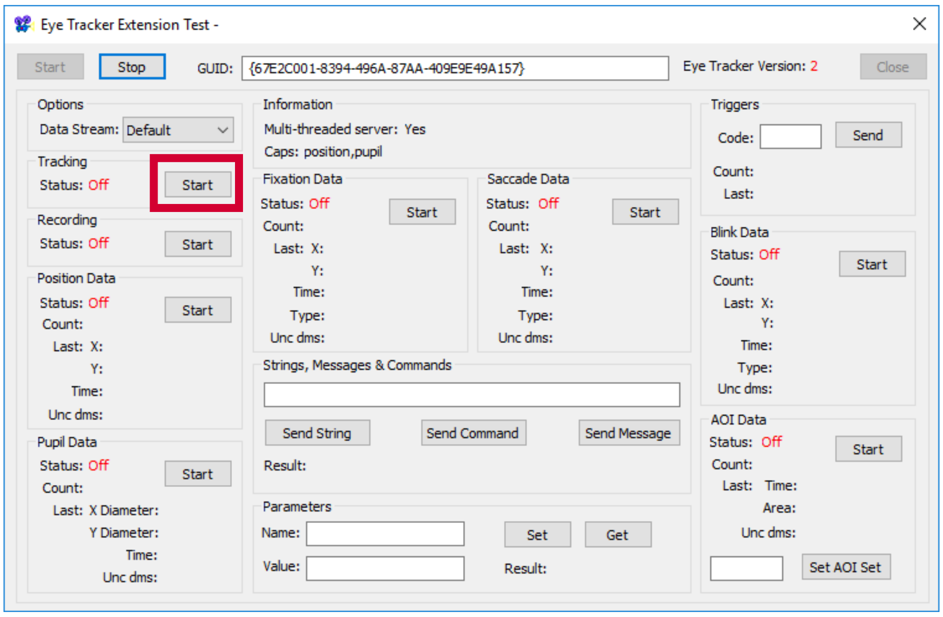
Caps: position,pupil

Status: Off

Name: uri

Value: tet-tcp://169.254.5.86

6. **Click Start** in the **Tracking** box to start tracking.



Eye Tracker Extension Test -

GUID: {67E2C001-8394-496A-87AA-409E9E49A157}

Eye Tracker Version: 2

Data Stream: Default

Multi-threaded server: Yes

Caps: position,pupil

Tracking Status: Off

Recording Status: Off

Position Data Status: Off

Pupil Data Status: Off

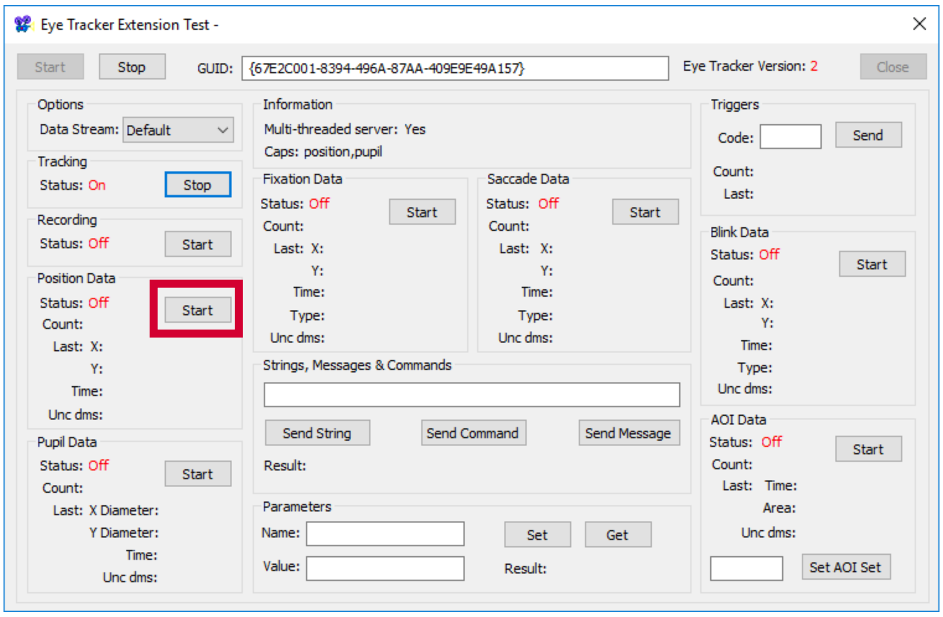
Fixation Data Status: Off

Saccade Data Status: Off

Blink Data Status: Off

AOI Data Status: Off

7. **Click Start** in the **Position Data** box to start getting eye position data.



Eye Tracker Extension Test -

GUID: {67E2C001-8394-496A-87AA-409E9E49A157}

Eye Tracker Version: 2

Data Stream: Default

Multi-threaded server: Yes

Caps: position,pupil

Tracking Status: On

Recording Status: Off

Position Data Status: Off

Pupil Data Status: Off

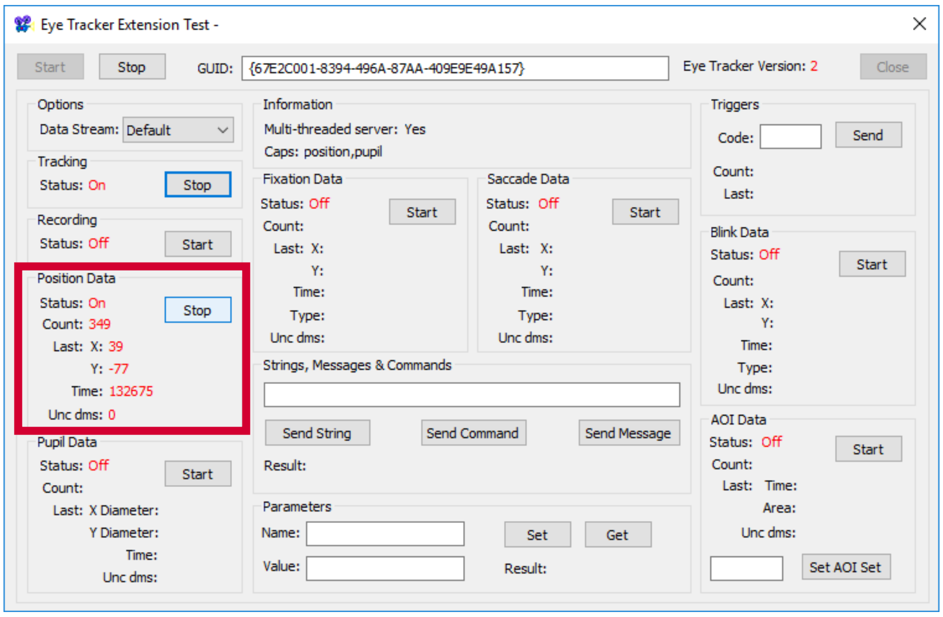
Fixation Data Status: Off

Saccade Data Status: Off

Blink Data Status: Off

AOI Data Status: Off

8. Eye position data should be reported in the **Position Data** box.



Eye Tracker Extension Test -

GUID: {67E2C001-8394-496A-87AA-409E9E49A157}

Eye Tracker Version: 2

Options
Data Stream: Default

Information
Multi-threaded server: Yes
Caps: position,pupil

Triggers
Code:
Count:
Last:

Tracking
Status: On

Fixation Data
Status: Off
Count:
Last: X:
Y:
Time:
Type:
Unc dms:

Saccade Data
Status: Off
Count:
Last: X:
Y:
Time:
Type:
Unc dms:

Recording
Status: Off

Blink Data
Status: Off
Count:
Last: X:
Y:
Time:
Type:
Unc dms:

Position Data
Status: On
Count: 349
Last: X: 39
Y: -77
Time: 132675
Unc dms: 0

Strings, Messages & Commands
Result:

AOI Data
Status: Off
Count:
Last: Time:
Area:
Unc dms:

Pupil Data
Status: Off
Count:
Last: X Diameter:
Y Diameter:
Time:
Unc dms:

Parameters
Name:
Value:
Result:

# 3 Using the Extension within Presentation Experiments

## 3.1 Overview

This chapter will illustrate how to use the Presentation Extension for Tobii Pro to create your own experiments.

You will need a computer with the Presentation software and the Presentation Extension for Tobii Pro already installed. If you have not installed Presentation, please complete the installation before continuing. If you have not installed Presentation Extension for Tobii Pro, please refer to the installation section of this manual.

While this document shows you the basic steps how to use Presentation with Tobii Pro eye trackers, it is assumed that you are familiar with setting up and using Presentation experiments, especially using PCL and scenario files.

Several samples that show how to use the extension for your own Presentation experiments are provided with the extension.

To make sure that the test person's eyes are recognized by the eye tracker before running your experiment, you can use the Tobii Pro Eye Tracker Browser provided by Tobii (https://www.tobiipro.com/product-listing/eye-tracker-manager/).

## 3.2 Adding Tobii support to a Presentation experiment

Adding Tobii Pro eye tracker support to a Presentation experiment primarily involves adding commands to connect to the eye tracker, commands starting and stopping the tracking as well as starting and stopping getting data from the eye tracker.

**Overview of the steps necessary**

- Creating the eye tracker extension object
- Set parameters
- Calibrate the eye tracker
- Specify the data type to get from the eye tracker
- Start the tracking
- Process the eye tracking data
- Stop the tracking

## 3.3 Creating the eye tracker extension object

You can create an eye_tracker object using the new operator. The argument in this expression can be the name you assigned to the extension when you registered it with Presentation.

```
eye_tracker tobii = new eye_tracker("Tobii_ET");
```

Alternatively, you can use the unique identification number for the extension. For the Presentation Extension for Tobii Pro it is **{67E2C001-8394-496A-87AA-409E9E49A157}**.

```
eye_tracker tobii = new eye_tracker("{67E2C001-8394-496A-87AA-409E9E49A157}");
```

## 3.4 Set parameters

You can set several parameters that control the extension. This is done using the **eye_tracker::set_parameter** method. See chapter 4.1 for a list of all supported parameters.

```
# Set the host address of the Tobii Pro eye tracker and connect to it
tobii.set_parameter("Uri", "tet-tcp://169.254.5.86");

# Report only valid data samples (Default is False)
string result = tobii.set_parameter("SendAllValues", "True");

# Value reported in case of SendAllValues has been set to true and
# the eye tracker reports invalid gaze data
string result = tobii.set_parameter("InvalidDataVaue", "-10000");

# Tells the extension that the presentation of the calibration points should
# be randomly shuffled (Default is False)
string result = tobii.set_parameter("ShuffleCalibrationPoints", "True");
```

## 3.5 Calibrate the eye tracker

The calibration can be done by using either a so called automatic calibration or a custom calibration scheme. See Presentations online documentation for details (https://www.neurobs.com/presentation/docs/index_html).

### Automatic Calibration

During an automatic calibration procedure, the extension has full control of the stimulus display. The method does not return until the procedure is finished. You may optionally use your own image for the calibration display by defining a picture stimulus named **et_calibration**. The extension controls the location of all picture parts in this picture stimulus during calibration.

An automatic calibration is done with the **eye_tracker::calibrate**() method. The extension supports 2-, 5- and 9-point calibration.

```
# Calibrate the eye tracker
#
# This example uses a 5-point calibration scheme.
# The first parameter specifies the delay between the presentation of the
# calibration stimulus and the eye tracker starting collecting calibration data.
tobii.calibrate(tobii.CALIBRATE_DEFAULT, 1000, 5, 0);
```

### Custom Calibration

For almost all purposes, the **eye_tracker::calibrate**() method described above will be sufficient. However, in some situations it might be necessary to use a more complex stimulus display during calibration. For example, a small animation with sound might be required when dealing with very small children. Such a display cannot be produced using the **eye_tracker::calibrate**() method, regardless of the capabilities of the extension. For this purpose, the eye tracker extension provides a calibration procedure during which the PCL program handles the display.

This procedure uses the following methods:

> **eye_tracker::start_calibration**
> **eye_tracker::stop_calibration**,
> **eye_tracker::get_calibration_point**(), **eye_tracker::accept_point**()

The following example uses an ellipse picture stimulus as calibration target. It's size will be animated during calibration.

**SDL:**

```
# Define an orange circle on gray background as calibration stimuli.
# It's size will be changed during calibration
picture {
    background_color = 37, 37, 37;
        ellipse_graphic {
        ellipse_height = 70;
        ellipse_width = 70;
        color = 227, 113, 0;
    } calib_circle;
    x = 0;
    y = 0;
} calibration_pic;
```

**PCL:**

```
# Calibrate the eye tracker using a custom calibration scheme.
#
# If an empty calibration point array is specified, the calibration
# uses a very simple 5-point calibration strategy.
# The eye parameter is ignored as Tobii just supports
# calibration for both eyes.
array<double> calibPoints[5][2] = {{0.0, 0.0}, {-768.0, -432.0}, {-768.0, 432.0},
{768.0, 432.0}, {768.0, -432.0}};

# Start calibration.
#
# The first parameter specifies the calibration coordinates to use.
# The second parameter has to be set to DEFAULT.
# The third parameter specifies the delay between the presentation of the
# calibration stimulus and the eye tracker starting collecting calibration data
# and has to be given as a string.
tobii.start_calibration(calibPoints, tobii.DEFAULT, "2000");

# Run until all calibration points have been shown.
loop
    int point = -1;
    int change_time = clock.time() + 40;
    int start_size = int(calib_circle.width());
    int size = start_size;
until
    point == 0 ||
    response_manager.last_response() == 2
begin
    double x;
    double y;
    int cpoint;

    tobii.get_calibration_point(cpoint, x, y);

    if(point != cpoint) then
        point = cpoint;

        if(point != 0) then
            calibration_pic.set_part_x(1, x);
            calibration_pic.set_part_y(1, y);
            size = start_size;
            calib_circle.set_dimensions(double(size), double(size));
            calib_circle.redraw();
            calibration_pic.present()
        end;
    end;

    if(clock.time() >= change_time) then
        change_time = change_time + 40;
        size = size - 2;

        if(size <= 10) then
            size = 10;
        end;

        calib_circle.set_dimensions(double(size), double(size));
        calib_circle.redraw();
        calibration_pic.present()
    end;
end;

# Stop calibration.
tobii.stop_calibration();
```

## 3.6 Specify the data type to get from the eye tracker

You must request to receive a particular type of data using the **eye_tracker::start_data**( ) method. This function can be called before or after tracking has been started. When the data is no longer needed you can stop receiving the particular type of data using the **eye_tracker::stop_data**( ) method.

```
# Getting eye tracker gaze position data
# without buffering
tobii.start_data(tobii.DATA_POSITION, false);
```

The extension currently supports the following data types:

- Gaze position data (**DATA_POSITION**)
- Pupil data (**DATA_PUPIL**)

The following tables show the information returned by the extension for the different data types. See Presentations online documentation for details (https://www.neurobs.com/presentation/docs/index_html).

| Gaze position data (DATA_POSITION) | | | |
|---|---|---|---|
| | Eye Argument | | |
| | **LEFT** | **RIGHT** | **DEFAULT** |
| **time [int]** | Timestamp of the associated position data in milliseconds converted to Presentation time. The timestamp of the first reported position data will be greater than zero because the eye trackers timer starts running as soon as the eye tracker object is created. | | |
| **x [double]** | x coordinate of the eye position in pixels or in custom units. | x coordinate of the eye position in pixels or in custom units. | x coordinate of the mean eye position (**(xl + xr) / 2)**) in pixels or in custom units. |
| **y [double]** | y coordinate of the eye position in pixels or in custom units. | y coordinate of the eye position in pixels or in custom units. | y coordinate of the mean eye position (**(yl + yr) / 2)**) in pixels or in custom units. |
| **uncertainty_dms [int]** | Maximum possible conversion error of the timestamp from the eye tracker's clock to Presentation's clock in tenths of milliseconds. | | |

| Pupil data (DATA_PUPIL) | | | |
|---|---|---|---|
| | Eye Argument | | |
| | LEFT | RIGHT | DEFAULT |
| time [int] | Timestamp of the associated pupil data in milliseconds converted to Presentation time. The timestamp of the first reported pupil data will be greater than zero because the eye trackers timer starts running as soon as the eye tracker object is created. | | |
| X_diameter [double] | Diameter of the subject's left eye pupil in millimetres. | Diameter of the subject's right eye pupil in millimetres. | Diameter of the subject's left eye pupil in millimetres. |
| Y_diameter [double] | | | Diameter of the subject's right eye pupil in millimetres. |
| uncertainty_dms [int] | Maximum possible conversion error of the timestamp from the eye tracker's clock to Presentation's clock in tenths of milliseconds. | | |

You can request that Presentation stores the incoming data in a buffer, or have it only maintain the most recent sample or event.

```
# Getting eye tracker gaze position data
# with buffering
tobii.start_data(tobii.DATA_POSITION, true);
```

If you buffer the data, you can access previous samples or events. Note that Presentation is not designed as a data acquisition system. Eye data received by Presentation is not saved to disk and does not appear in the logfile. You should only transmit data to Presentation if the stimulus display or scenario behavior is dependent on that data. Presentation uses circular buffers to store eye tracking data. If the buffer is filled, newer data will start to overwrite the data at the beginning of the buffer. Presentation keeps track of the total number of events received as well as the current position in the data buffer. Each type of data has its own buffer.

The buffer size to store the data can be set by using the **set_max_buffer_size**() function of the **eye_tracker** object. The buffer size is set to **100** by default and can be set to a value between **1** and **100000**.

```
# Set the buffer size for position data to 1000
int buffer_size = 1000;
tobii.set_max_buffer_size(tobii.DATA_POSITION, buffer_size);
```

## 3.7 Start the tracking

Tracking can be started through the extension using the **`eye_tracker::start_tracking()`** method. The extension will not automatically send eye tracking data to Presentation when it is tracking. You must request to receive a particular type of data using the **`eye_tracker::start_data()`** function. See Section 3.4 for details.

```
# Start the tracking.
tobii.start_tracking();
```

The extension supports simultaneous data from multiple eyes, or data of a specific type. Handling data for multiple data streams is essentially the same as for a single data stream. All data access methods of the **`eye_tracker`** object have an optional **eye** argument that you can use to refer to a particular set of data.

```
# Start getting gaze position data from the left eye without buffering.
tobii.start_data(tobii.LEFT, tobii.DATA_POSITION, false);
```

You may still use the methods without the **eye** argument if you specify the default data set to use by using the **`eye_tracker::set_default_data_set()`** method. By default the extension uses the **`DEFAULT`** data set which represents both eyes.

```
# Set data from the right eye as the default data set to use
tobii.set_default_data_set(tobii.RIGHT);

# Start getting gaze position data from default (i.e. right) data set
# without buffering.
tobii.start_data(tobii.DATA_POSITION, false);
```

## 3.8 Process the eye tracking data

After starting the tracking the eye tracking data requested can be processed within a loop.

```
# Run for 5 seconds and process unbuffered position data
loop
    int end_time = clock.time() + 5000;
until
    clock.time() >= end_time
begin
    if(tobii.new_position_data() > 0) then
        eye_position_data posData = tobii.last_position_data();

        # Print the position data coordinates to the terminal
        term.print(int(posData.x()));
        term.print("    ");
        term.print(int(posData.y()));
        term.print("    ");
        term.print_line(posData.time());
    end;
end;
```

Buffered data can be processed as shown in the following example.

```
# Run for 5 seconds and process buffered position data
loop
    int position = 0;
    int end_time = clock.time() + 5000;
until
    clock.time() >= end_time
begin
    loop
    until
        position = tobii.buffer_position(DATA_POSITION);
    begin
        position = position + 1;
        if(position > buffer_size) then
            position = 1
        end;
        eye_position_data posData = tracker.get_position_data(position);

        # Print the position data coordinates to the terminal
        term.print(int(posData.x()));
        term.print("    ");
        term.print(int(posData.y()));
        term.print("    ");
        term.print_line(posData.time());
    end;
    # Perform other tasks here
end;
```

## 3.9 Stop the tracking

Tracking should be stopped through the extension using the **`eye_tracker::stop_tracking`**`()` method. Do not forget to stop the tracking at the end of your experiment as this will do necessary clean up within the extension.

```
# Stop the tracking.
tobii.stop_tracking();
```

# 4 Reference Guide

This section describes the functions, parameters and commands that are supported by the extension.

## 4.1 Functions supported by the Extension

The extension supports a subset of the functions of Presentations **`eye_tracker`** object. See Presentations online documentation for details (https://www.neurobs.com/presentation/docs/index_html).

**List of the supported functions**

- **`new eye_tracker`**`(` **`string`** `object_id ) :` **`eye_tracker`**
- **`buffer_position`**`( [`**`int`** `eye,]` **`int`** `data_type ) :` **`int`**
- **`calibrate`**`(` **`int`** `calibration_type,` **`double`** `parameter1,` **`double`** `parameter2,` **`double`** `parameter3 ) :` **`string`**
- **`clear_buffer`**`( [`**`int`** `eye,]` **`int`** `data_type ) :` **`void`**
- **`event_count`**`( [`**`int`** `eye,]` **`int`** `data_type ) :` **`int`**
- **`get_calibration_point`**`(` **`int`**`& index,` **`double`**`& x,` **`double`**`& y ) :` **`string`**
- **`get_parameter`**`(` **`string`** `name ) :` **`string`**
- **`get_position_data`**`( [`**`int`** `eye,]` **`int`** `index ) :` **`eye_position_data`**
- **`get_pupil_data`**`( [`**`int`** `eye,]` **`int`** `index ) :` **`pupil_data`**
- **`get_status`**`( ) :` **`int`**
- **`last_position_data`**`( [`**`int`** `eye] ) :` **`eye_position_data`**
- **`last_pupil_data`**`( [`**`int`** `eye] ) :` **`pupil_data`**
- **`new_position_data`**`( [`**`int`** `eye] ) :` **`int`**
- **`new_pupil_data`**`( [`**`int`** `eye] ) :` **`int`**
- **`send_command`**`(` **`string`** `message ) :` **`int`**
- **`set_abort_on_error`**`(` **`bool`** `abort ) :` **`void`**
- **`set_default_data_set`**`(` **`int`** `eye ) :` **`void`**
- **`set_max_buffer_size`**`( [`**`int`** `eye,]` **`int`** `data_type,` **`int`** `size ) :` **`void`**
- **`set_parameter`**`(` **`string`** `name,` **`string`** `value ) :` **`string`**
- **`start_calibration`**`(` **`array<double`**`,2> points,` **`int`** `eye,` **`string`** `parameters ) :` **`string`**
- **`start_data`**`(` **`int`** `data_type [,` **`bool`** `store_data] ) :` **`string`**

- **`start_data`**`(` **`int`** `eye,` **`int`** `data_type,` **`bool`** `store_data ) :` **`string`**
- **`start_tracking`**`( ) :` **`string`**
- **`stop_calibration`**`( ) :` **`string`**
- **`stop_data`**`( [`**`int`** `eye,]` **`int`** `data_type ) :` **`string`**
- **`stop_tracking`**`( ) :` **`string`**
- **`supports`**`(` **`int`** `feature_code ) :` **`bool`**
- **`version`**`( ) :` **`int`**

The following sections give detailed information about functions that have special parameters or meaning that is not covered in Presentations documentation.

## 4.2 Automatic Calibration

To perform an automatic calibration in your experiment use the **`eye_tracker::calibrate`** PCL function.

### *4.2.1 eye_tracker::calibrate*

**Syntax**

```
calibrate( int calibration_type, double parameter1, double
parameter2, double parameter3 ) : string
```

**Description**

This method instructs the eye tracker to perform a calibration operation. The method does not return until the calibration procedure has completed. If you define a picture stimulus named `et_calibration`, this picture stimulus will be used for the calibration display. The extension controls the positioning of the picture parts in this stimulus during calibration. If you don't define a picture stimulus named `et_calibration`, Presentation will use a default white cross on a black background.

The order in which the calibration points are shown can be randomly shuffled. See section 4.4.4 for details.

**Parameters**

`int calibration_type`

Not used, should be set to `CALIBRATE_DEFAULT`.

`double parameter1`

Delay time in milliseconds that the extension waits between presenting the calibration point and starting to collect calibration data.

`double parameter2`

Number of calibration points to use. This can be either `2`, `5` or `9`. The default value is `5`.

`double parameter3`

Not used, should be set to `0`.

**Return Value**

An empty string or an error description if there was an error.

**Example**

```
string result = tobii.calibrate( tobii.CALIBRATE_DEFAULT, 1000,
5, 0 );
```

## 4.3 Custom Calibration

In some situations, it might be necessary to use a more complex stimulus display during calibration than possible with the automatic calibration. For example, a small animation with sound might be required. For this purpose, you may provide a calibration procedure during which your PCL program handles the display. Use the **`eye_tracker::start_calibration`**, **`eye_tracker::stop_calibration`**, and **`eye_tracker::get_calibration_point`** methods for this purpose.

### *4.3.1 eye_tracker::start_calibration*

**Syntax**

```
start_calibration( array<double,2> points, int eye, string
parameter ) : string
```

**Description**

This method instructs the eye tracker to start a custom calibration procedure in which the scenario will control the stimulus display. The `points` argument is a two-dimensional array that defines the specific calibration points to be used. Each element of the array is an array of size 2 containing the x and y coordinates of the calibration points. The value is in pixels.

The order in which the calibration points are shown can be randomly shuffled. See section 4.4.4 for details.

**Parameters**

`array<double,2> points`

Two-dimensional array that defines the specific calibration points to be used. Each element of the array is an array of size 2 containing the x and y coordinates of the calibration points. The value is in pixels.

If an empty array is specified, a default 5-point scheme will be.

`int eye`

Ignored, because only binocular calibration is supported at the moment. Should be set to `DEFAULT`.

`string parameter`

Delay time in milliseconds that the extension waits between presenting the calibration point and starting to collect calibration data.

**Return Value**

An empty string or an error description if there was an error.

**Example**

```
array<double> calibPoints[5][2] = {{0.0, 0.0},
    {-768.0, -432.0}, {-768.0, 432.0}, {768.0, 432.0},
    {768.0, -432.0}};
string result = tobii.start_calibration( calibPoints,
    tobii.DEFAULT,  "2000" );
```

### 4.3.2 *eye_tracker::stop_calibration*

**Syntax**

```
stop_calibration( ) : string
```

**Description**

This method instructs the eye tracker to stop the calibration procedure.

**Return Value**

An empty string or an error description if there was an error.

**Example**

```
string result = tobii.stop_calibration( );
```

### 4.3.3 *eye_tracker::get_calibration_point*

**Syntax**

```
get_calibration_point( int& index, double& x, double& y ) :
string
```

**Description**

This method is used to obtain the location of the current calibration point.

The calibration procedure can move to the next point at any time, so you should poll this method continuously during the calibration display. The `index` argument is set to the index of the current calibration point, where the first calibration point has index 1. If the calibration has ended, 0 is returned in this argument. Note that you should still call **`eye_tracker::stop_calibration`** even after receiving 0 for the calibration point index. The x and y arguments are set to the coordinates of the calibration point. These values will be in pixels.

**Parameters**

`int& index`

Set to the index of the current calibration point, where the first calibration point has index 1.

`double& x`

x coordinate of the current calibration point in pixels.

`double& y`

y coordinate of the current calibration point in pixels.

**Return Value**

An empty string or an error description if there was an error.

**Example**

```
int calibPoint;
double x;
double y;
string result = tobii.get_calibration_point( calibPoint,
    x,  y );
```

## 4.4 Spported Parameters

Parameters are set by using the **`eye_tracker::set_parameter`** function. To retrieve the actual value of a parameter use the **`eye_tracker::get_parameter`** function.

Parameter names and values are case-insensitive.

The following sections describe the parameters that are supported by the extension

### *4.4.1 Uri Parameter*

**Syntax**

```
string result = set_parameter( "Uri",
    "tet-tcp://XXX.XXX.XXX.XXX" );
```

```
string uri = get_parameter( "Uri" );
```

**Description**

This parameter is used to set the address of a specific eye tracker to connect to or to get the address of the currently connected eye tracker.

By default, the extension connects to the first connected eye tracker found. By setting a specific address of an eye tracker with the uri parameter, the extension tries to connect to that eye tracker.

The address of any connected eye tracker can be found using the Tobii Pro Eye Tracker Manager tool.

**Default Value**

The address of the first connected eye tracker found.

**Return Value**

An empty string or an error description if there was an error when setting the parameter.

The address of the currently connected eye tracker when getting the parameter.

**Example**

```
string result = set_parameter( "Uri",
    "tet-tcp://169.254.5.86" );
```

```
string uri = get_parameter( "Uri" );
```

### *4.4.2 SendAllValues  Parameter*

**Syntax**

```
string result = set_parameter( "SendAllValues",
    ["True"|"False"] );
```

```
string value = get_parameter( "SendAllValues" );
```

**Description**

This parameter is used to tell the extension how it should handle invalid gaze data that is reported by the eye tracker. The possible values to set can be be `"True"` or `"False"`.

If set to `"True"`, the extension reports all data it gets from the eye tracker. Invalid gaze data values will be set to the value specified by the InvalidDataValue parameter. See also section 4.4.3.

If set to `"False"`, the extension will only report valid gaze data it receives from the eye tracker.

**Default Value**

By default, this parameter is set to false.

**Return Value**

An empty string or an error description if there was an error when setting the parameter.

The currently set value when getting the parameter.

**Example**

```
string result = set_parameter( "SendAllValues", "True" );
```

```
string value = get_parameter( "SendAllValues" );
```

### *4.4.3 InvalidDataValue Parameter*

**Syntax**

```
string result = set_parameter( "InvalidDataValue",
    "[value]" );
```

```
string value = get_parameter( "InvalidDataValue" );
```

**Description**

This parameter is used to set the data value that the extension should report in case of invalid gaze data. See also section 4.4.2.

`value` can only be an integer value (e.g. -10000). Floating point values are not supported.

**Default Value**

By default, this parameter is set to -10000.

**Return Value**

An empty string or an error description if there was an error when setting the parameter.

The currently set value as a string when getting the parameter.

**Example**

```
string result = set_parameter( "InvalidDataValue", "-10000" );
```

```
string value = get_parameter( "InvalidDataValue" );
```

### *4.4.4 ShuffleCalibrationPoints Parameter*

**Syntax**

```
string result = set_parameter( "ShuffleCalibrationPoints",
    ["True"|"False"] );
string value = get_parameter( "ShuffleCalibrationPoints" );
```

**Description**

This parameter is used to tell the extension if it should randomly shuffle the array of calibration points used during an automatic or custom calibration. The possible values to set can be be `"True"` or `"False"`.

If set to `"True"`, the extension will randomly shuffle the presentation of the calibration points.

If set to `"False"`, the calibration points will be presented in a predefined, and in case of automatic calibration also hardcoded, order.

**Default Value**

By default, this parameter is set to false.

**Return Value**

An empty string or an error description if there was an error when setting the parameter.

The currently set value when getting the parameter.

**Example**

```
string result = set_parameter( "ShuffleCalibrationPoints",
    "True" );
string value = get_parameter( "ShuffleCalibrationPoints" );
```

#### *4.4.5 Version Parameter*

**Syntax**

```
string value = get_parameter( "Version" );
```

**Description**

This parameter is used to get the current version of the extension in the form of Major.Minor.Revision (e.g. 3.0.0).

Note that this has nothing to do with the version that is reported by the **`eye_tracker::version`** function which reports the interface version that the extension uses (i.e. 2).

**Return Value**

The current version of the extension as a string in the form of Major.Minor.Revision.

**Example**

```
string value = get_parameter( "Version" );
```

## 4.5 Supported Commands

Commands are sent do the extension using the **`eye_tracker::send_command`** function.

Command names are case-insensitive.

Note that -1 will be returned as return value if an empty or unknown command is provided.

The following sections describe the commands that are supported by the extension

### 4.5.1 SaveCalibration Command

**Syntax**

```
int result = send_command( "SaveCalibration [filename]" );
```

**Description**

Saves the current active calibration in use by the eye tracker to a file. If a file with the given path and filename exists, it will be overwritten.

**Parameters**

The absolute path and filename of the calibration file to store. Make sure to escape the backslash character using a double backslash (`\\`).

**Return Value**

0 on success, -1 on error.

**Example**

```
int result = send_command( "SaveCalibration C:\\user1.cal" );
```

### 4.5.2 LoadCalibration Command

**Syntax**

```
int result = send_command( "LoadCalibration [filename]" );
```

**Description**

Loads a calibration stored in a file and sets it as the active calibration to be used by the eye tracker. Any previous active calibration in use is overwritten.

**Parameters**

The absolute path and filename of the calibration file to store. Make sure to escape the backslash character using a double backslash (`\\`).

**Return Value**

0 on success, -1 on error.

**Example**

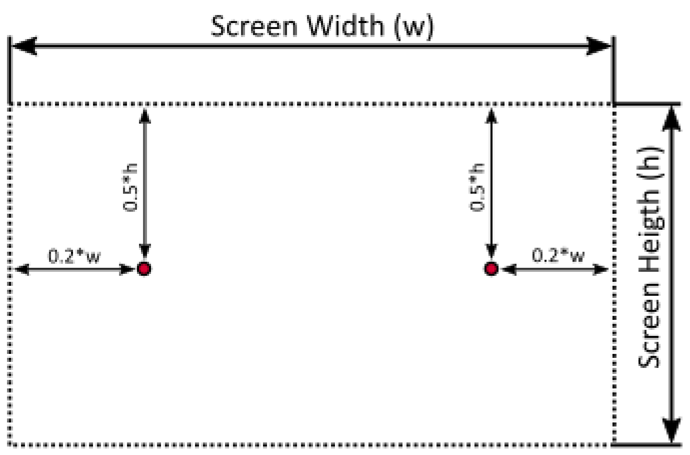
```
int result = send_command( "LoadCalibration C:\\user1.cal" );
```

# 5 Appendixes

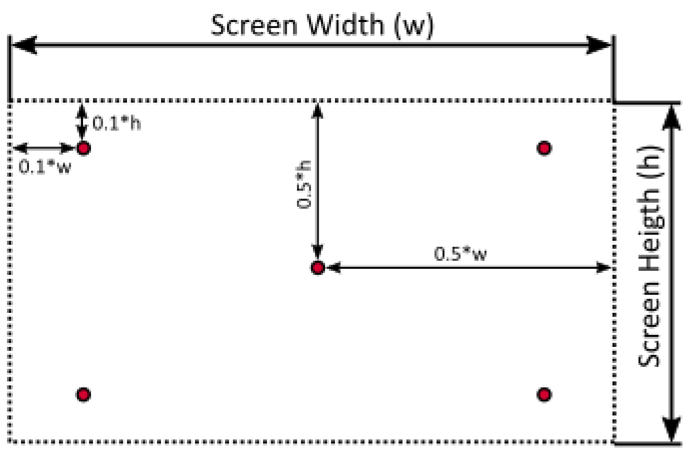
## 5.1 Appendix A – Calibration Grids

The default calibration grids used by the automatic calibration for the different number of calibration points. Note that the custom calibration also uses the 5-point scheme when an empty calibration points array is specified.

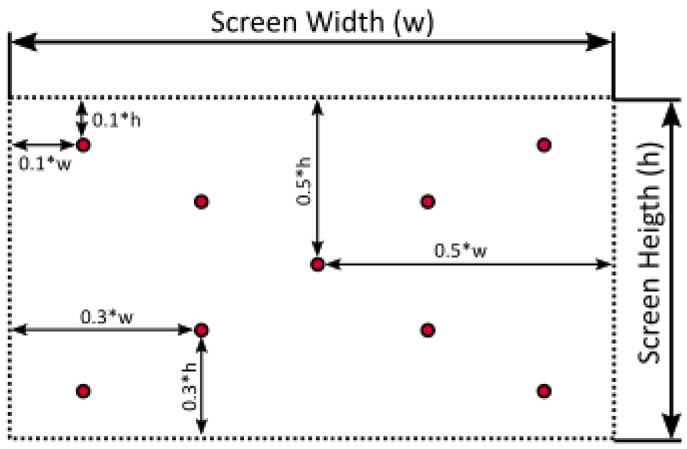
**2-Points**



**5-Points**



### 9-Points



## 5.2 Appendix B – Release Notes

### Version 3.0.1

- Initial release