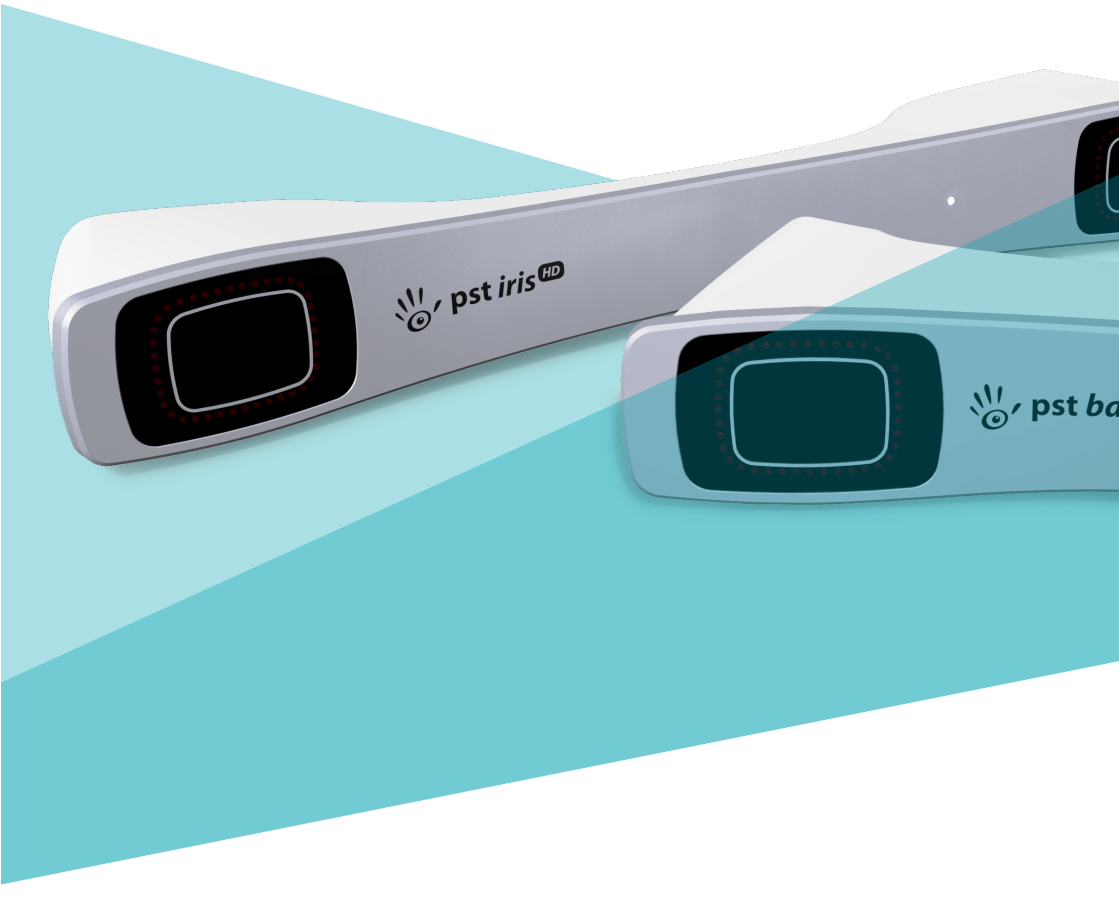
ps-tech

PST

Classic SDK Manual



PS-Tech B.V.
Falckstraat 53 hs
NL 1017VV Amsterdam
The Netherlands
Call: +31 (0)20 331 1214
Fax : +31 (0)20 524 8797

info@ps-tech.com
http://www.ps-tech.com

Rev. 1.2.3-0-g357a84f

Legal

License agreement

The products of PS-Tech B.V. (PS-Tech) come with a software license agreement. This END USER LICENSE AGREEMENT (EULA) is shipped with each product, and is available on request at the offices of PS-Tech B.V.

In no event shall PS-Tech be held liable for any incidental, indirect, or consequential damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use the software or hardware.

Patent liability

No patent liability is assumed with respect to the use of the products of PS-Tech B.V.

Copyright information

Portions of the software included in this package contain licensed third-party technology. With some of these, you also may have additional rights, particularly to receive source code of these projects. The LDL and COLAMD libraries of the SuiteSparse project are licensed under the GNU LGPL. The SSBA library is licensed under the GNU LGPL. This software is based in part on the work of the FLTK project. The DevIL library is licensed under the GNU LGPL. The relevant third-party licenses are included in the license.txt file in your PST installation. Please contact PS-Tech to obtain source copies of these libraries.

Contents

1 Software development kit

The Classic PST Software Development Kit (Classic SDK) provides an interface between the PST tracking system and your own software applications.

Note that with the release of the new PST SDK in version 5.0.0 of the PST software package, the Classic PST SDK is labeled as legacy software and is only offered for backwards compatibility. When a new project making use of the PST tracking system is started it is highly recommended to use the new PST SDK. Documentation for the new PST SDK can be found in the Start menu as "PST SDK Manual" or can be opened by opening the "index.html" file in the "Development/docs" directory in the installation path.

1.1 Usage

To use the Classic PST SDK in your own software, include the header file "pstapi.h" in your project. The Classic PST SDK library is dynamically (pst.lib/pst.dll or pst.so) or statically (pst.a) linked with your program.

Note that the Classic PST SDK communicates with the PST client software that is included with your PST installation. If this application is not running you will not receive tracker events in your application, even if the tracker unit itself is running.

The Classic PST SDK contains two data types to describe tracker data events: `PSTSensor` and `PSTPoint`.

1.2 Datatype: PSTSensor

Description
PSTSensor sensor events are generated when a tracking target is visible and has been identified by the PST.

Member documentation

name	`char[80]`	Name of the tracking target as listed in the PST client software.
id	`int`	Identifier of the tracking target as listed in the PST client software.
pose	`float[16]`	Row-major 4 $\times$ 4 transformation matrix describing the pose of the tracking target in the coordinate system as defined in the PST client software (see the "Reference coordinate system" Section in the PST Manual). The pose is defined as: $\begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \\ p_{12} & p_{13} & p_{14} & p_{15} \end{bmatrix} = \begin{bmatrix} U_x & V_x & W_x & T_x \\ U_y & V_y & W_y & T_y \\ U_z & V_z & W_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ where p_i represents the elements from the pose, the vectors U, V, W represent the 3 $\times$ 3 rotation matrix in radians, and T represents the translation vector in meters.
timestamp	`double`	Timestamp of the moment the cameras captured the data. The timestamp uses the system clock provided in seconds since system boot (Windows) or Epoch (Linux).

1.3 Datatype: PSTPoint

Description
Point events are generated for single visible 3D points that have not been identified as part of an tracking target.

Member documentation

id	`int`	Identifier of the 3D point. As a single 3D point has no features to distinguish it from another, points are given an identifier based on their previous motion. Note that there is no guarantee that the identifier is consistent between sensor updates.
pos	`float[3]`	The 3D position of the point in meters.
timestamp	`double`	Timestamp of the moment the cameras captured the data. The timestamp uses the system clock provided in seconds since system boot (Windows) or Epoch (Linux).

1.4 Header pstapi.h

Description
The interface to the PST client software.

Function documentation

`int pst_connect()`
Connect to the PST
Return value `int` One on success, zero on failure

`int pst_disconnect()`
Disconnect from the PST
Return value `int` One on success, zero on failure

`int pst_sensor_changed()`
Check if any PST sensor has been updated since the last time it was read by the SDK
Parameters `id` The identifier of the device (0-99)
Return value `int` One if new data is available, zero if no new data is available

`int pst_sensor_changed_by_id(int id)`
Check if the PST sensor indicated by `id` has been updated since the last time it was read by the SDK
Parameters `id` The identifier of the device (0-99)
Return value `int` One if new data is available, zero if no new data is available

`int pst_get_sensor(struct PSTSensor* sensor)`
Get the last PST sensor event if a new event is available

Parameters	`sensor`	A pointer to an allocated PSTSensor struct to receive a new event
Return value	`int`	One if a new event is returned, zero if no new data is available

`int pst_get_sensor_by_id(int id, struct PSTSensor* sensor)`
Get the last PST sensor event with the given id if a new event is available

Parameters	`id`	The identifier of the device (0-99)
	`sensor`	A pointer to an allocated `PSTSensor` struct to receive a new event
Return value	`int`	One if new data is available, zero if no new data is available

`int pst_point_changed()`
Check if any PST point has been updated since the last time it was read by the SDK

Return value	`int`	One if a new point is available, zero if no new point is available

`int pst_get_point(struct PSTPoint* point)`
Get the last PST point event if a new event is available

Parameters	`id`	A pointer to an allocated PSTPoint struct to receive a new event
Return value	`int`	One if a new point is available, zero if no new point is available

`int pst_get_connection_state(int* state)`
Get the connection state of the SDK to the PST client

Parameters	`state`	A pointer to an int. After the call returns successfully, `state` will be set to one if a connection is active, zero otherwise.
Return value	`int`	One if the state was received successfully, zero otherwise

1.5 Example

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "pstapi.h"
```

```
int main(int argc, char **argv)
{
    int i, j;
    struct PSTSensor sensor;

    // connect to the PST
    if (!pst_connect())
        exit(1);

    // infinite loop...
    while (1)
    {
        // loop over all new sensor events
        while (pst_get_sensor(&sensor))
        {
            // print out the name and id
            printf("Device: \"%s\", id: %d\n",
              sensor.name, sensor.id);

            // print the rotation matrix
            printf(" Orientation:\n");
            for (i = 0; i < 3; ++i)
            {
                printf(" ");
                for (j = 0; j < 3; ++j)
                    printf("%.2f ",
                      sensor.pose[i * 4 + j]);
                printf("\n");
            }

            // print the translation vector
            printf("\n Translation:\n ");
            for (i = 0; i < 3; ++i)
                printf("%.2f ",
                  sensor.pose[i * 4 + 3]);
            printf("\n\n");
        }
    }

    // disconnect from the PST
```

```
    pst_disconnect();

    return 0;
}
```