INTEL® PERCEPTUAL COMPUTING SDK

Reference Manual

3rd Party Framework and Game Engine Support

API Version 1.0

LEGAL DISCLAIMER

THIS DOCUMENT CONTAINS INFORMATION ON PRODUCTS IN THE DESIGN PHASE OF DEVELOPMENT.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

INTEL MAY MAKE CHANGES TO SPECIFICATIONS AND PRODUCT DESCRIPTIONS AT ANY TIME, WITHOUT NOTICE. DESIGNERS MUST NOT RELY ON THE ABSENCE OR CHARACTERISTICS OF ANY FEATURES OR INSTRUCTIONS MARKED "RESERVED" OR "UNDEFINED." INTEL RESERVES THESE FOR FUTURE DEFINITION AND SHALL HAVE NO RESPONSIBILITY WHATSOEVER FOR CONFLICTS OR INCOMPATIBILITIES ARISING FROM FUTURE CHANGES TO THEM. THE INFORMATION HERE IS SUBJECT TO CHANGE WITHOUT NOTICE. DO NOT FINALIZE A DESIGN WITH THIS INFORMATION.

THE PRODUCTS DESCRIBED IN THIS DOCUMENT MAY CONTAIN DESIGN DEFECTS OR ERRORS KNOWN AS ERRATA WHICH MAY CAUSE THE PRODUCT TO DEVIATE FROM PUBLISHED SPECIFICATIONS. CURRENT CHARACTERIZED ERRATA ARE AVAILABLE ON REQUEST.

CONTACT YOUR LOCAL INTEL SALES OFFICE OR YOUR DISTRIBUTOR TO OBTAIN THE LATEST SPECIFICATIONS AND BEFORE PLACING YOUR PRODUCT ORDER.

COPIES OF DOCUMENTS WHICH HAVE AN ORDER NUMBER AND ARE REFERENCED IN THIS DOCUMENT, OR OTHER INTEL LITERATURE, MAY BE OBTAINED BY CALLING 1-800-548-4725, OR BY VISITING INTEL'S WEB SITE HTTP://WWW.INTEL.COM.

ANY SOFTWARE SOURCE CODE REPRINTED IN THIS DOCUMENT IS FURNISHED UNDER A SOFTWARE LICENSE AND MAY ONLY BE USED OR COPIED IN ACCORDANCE WITH THE TERMS OF THAT LICENSE ANY SOFTWARE SOURCE CODE REPRINTED IN THIS DOCUMENT IS FURNISHED UNDER A SOFTWARE LICENSE AND MAY ONLY BE USED OR COPIED IN ACCORDANCE WITH THE TERMS OF THAT LICENSE

INTEL, THE INTEL LOGO, INTEL CORE, INTEL MEDIA SOFTWARE DEVELOPMENT KIT (INTEL MEDIA SDK) ARE TRADEMARKS OR REGISTERED TRADEMARKS OF INTEL CORPORATION OR ITS SUBSIDIARIES IN THE UNITED STATES AND OTHER COUNTRIES.

MPEG IS AN INTERNATIONAL STANDARD FOR VIDEO COMPRESSION/DECOMPRESSION PROMOTED BY ISO. IMPLEMENTATIONS OF MPEG CODECS, OR MPEG ENABLED PLATFORMS MAY REQUIRE LICENSES FROM VARIOUS ENTITIES, INCLUDING INTEL CORPORATION.

*OTHER NAMES AND BRANDS MAY BE CLAIMED AS THE PROPERTY OF OTHERS.

COPYRIGHT © 2010-2013, INTEL CORPORATION. ALL RIGHTS RESERVED.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Table of Contents

Intel® Perceptual Computing SDK 1

Document Conventions 1

Acronyms and Abbreviations 1

Programming Guide 2

Unity Setup and Sample 2

Processing Setup and Sample 5

openFrameworks Setup and Sample 7

Interface and Function Reference 9

AcquireFrame 9

Close 10

Init 10

IsDisconnected 11

PauseFrame 11

QueryDepthMap 12

QueryDepthMapSize 12

QueryDeviceProperty 13

QueryFaceID 13

QueryFaceLandmarkData 14

QueryFaceLandmarkData[] 15

QueryFaceLandmarkPose 15

QueryFaceLocationData 16

QueryGeoNode 17
QueryGesture 17
QueryIRMap 18
QueryIRMapSize 19
QueryLabelMap 19
QueryLabelMapAsImage 20
QueryLabelMapSize 20
QueryRGB 21
QueryRGBSize 22
QueryUVMap 22
QueryUVMapSize 23
QueryVoiceRecognized 23
ReleaseFrame 24
SetDeviceProperty 24
SetVoiceCommands 25
SetVoiceDictation 25
Enumerator Reference 27
Mode 27

Intel® Perceptual Computing SDK

The Intel® Perceptual Computing SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The library's purpose is to lower the barriers to using these algorithms and shift the application developers' focus from coding the algorithm details to innovating on the usage of these algorithms for next generation human computer experience.

This document describes the 3rd party framework and game engine support of the Intel® Perceptual Computing SDK Application Programming Interface (API). The other reference manuals released in the SDK describe different algorithms and their API definitions.

Document Conventions

The SDK API uses the Verdana typeface for normal prose. With the exception of section headings and the table of contents, all code-related items appear in the `Courier New` typeface (`pxcStatus`). Hyperlinks appear in underlined boldface, such as **`pxcStatus`**.

Acronyms and Abbreviations

API	Application Programming Interface
SDK	Software Development Kit

Programming Guide

The Intel® Perceptual Computing SDK provides experimental support of the following frameworks and game engines:

- The Processing* framework version 1.5.1.
- The openFrameworks* version 0071.
- The Unity* game engine (professional) version 3.5.1f2.

With the exception of the openFrameworks, which supports SDK functions or utilities natively, the support is based on c-library **`libpxcupipeline`**, a customization of the **`UtilPipeline`** utility class, which enables easy access to the following features:

- Color picture capturing
- Face location and landmark detection/tracking
- Hand/finger tracking and pose/gesture recognition

The general programming procedures are similar for the supported frameworks and game engines since the support is based on the same underlying library:

- **Initialization**:

  The application uses the **`Init`** function to initialize the pipeline, which can be a combination of color, face, and gesture processing.

- **Data processing**:

  The **`AcquireFrame`** function waits for the current frame processing to complete and locks the processing results, and the **`ReleaseFrame`** function to release the lock and prepare for the next frame processing.
  In between the **`AcquireFrame`** function and the **`ReleaseFrame`** function, the application can call the **`Query`** series of functions to retrieve color, face, and gesture processing results.

- **Closing down**:

  The application uses the **`Close`** function to close the pipeline and free all resources.

Unity Setup and Sample

The Unity environment must be prepared as follows:

- Under Assets, create a directory called **Plugins**.
- Copy the following files from **framework\Unity\hellounity\Assets\Plugins**
  - **libpxcupipeline.dll**
  - **pxcm-structures.cs**
  - **pxcupipeline.cs**

Example 1 shows a simple Unity application, which renders the label map image onto some object's **mainTexture** element. The screen shot is shown in Figure 1. The sample calls the **Init** function in the **Start()** function, the **Close** function in the **OnDisable()** function, and process data in the **Update**() function. The sample uses the **QueryLabelMapAsImage** function to retrieve the label map image and then draws to the object's main texture.

```
using UnityEngine;
using System;
using System.Runtime.InteropServices;

public class TexturePlayback : MonoBehaviour {
	private Color[] px;
	private Texture2D tt;
	private PXCUPipeline pp;
	void Start () {
		pp=new PXCUPipeline();
		pp.Init(PXCUPipeline.Mode.GESTURE);
		int[] size=new int[2];
		pp.QueryLabelMapSize(ref size);
		tt=new Texture2D(size[0],size[1],TextureFormat.ARGB32,false);
		renderer.material.mainTexture=tt;
	}
	void OnDisable() {
		pp.Close();
	}
	void Update () {
		if (!pp.AcquireFrame(false)) return;
		if (pp.QueryLabelMapAsImage(tt)) tt.Apply();
		pp.ReleaseFrame();
	}
}
```

Example 1: A Simple Unity* Application

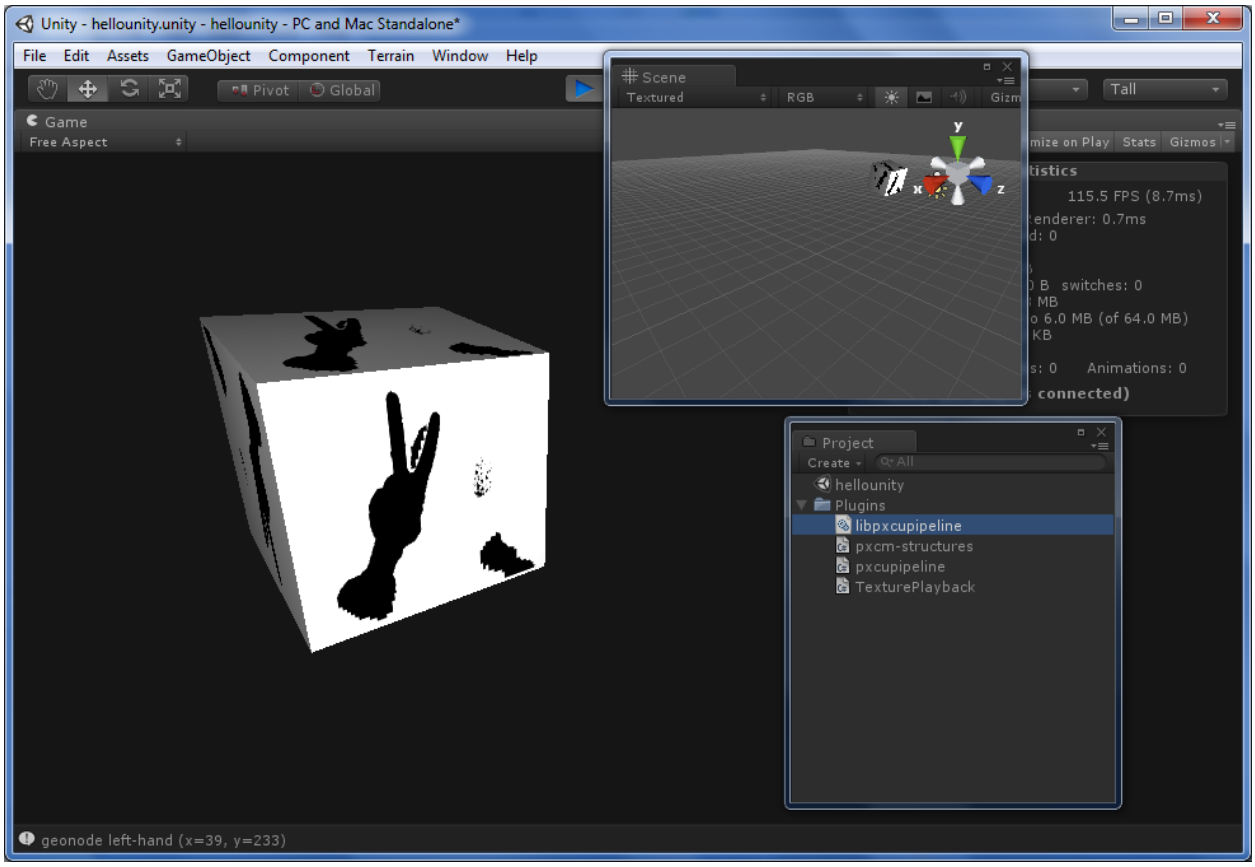

Figure 1: The Simple Unity* Application Screen Shoot

Processing Setup and Sample

The Processing framework environment can be prepared by copying the **`framework\Processing\libraries`** directory and everything underneath to your sketch directory.

Example 2 shows a simple Processing application, which renders the label map image onto the canvas. The screen shot is shown in Figure 2. The sample calls the **`Init`** function in the **`Setup()`** function, and processes data in the **`Draw`**() function. The sample uses the **`QueryLabelMapAsImage`** function to retrieve the label map image and then draws to the canvas. The **`Close`** function is never used.

```
import intel.pcsdk.*;
private PImage display;
private PXCUPipeline pp;
void setup() {
       pp=new PXCUPipeline(this);
       pp.Init(PXCUPipeline.GESTURE);
       int[] size=new int[2];
       pp.QueryLabelMapSize(size);
       display=createImage(size[0],size[1],RGB);
}
void draw() {
       if (!pp.AcquireFrame(false)) return;
       if (pp.QueryLabelMapAsImage(display)) image(display,0,0);
       pp.ReleaseFrame();
}
```

Example 2: A Simple Processing* Application

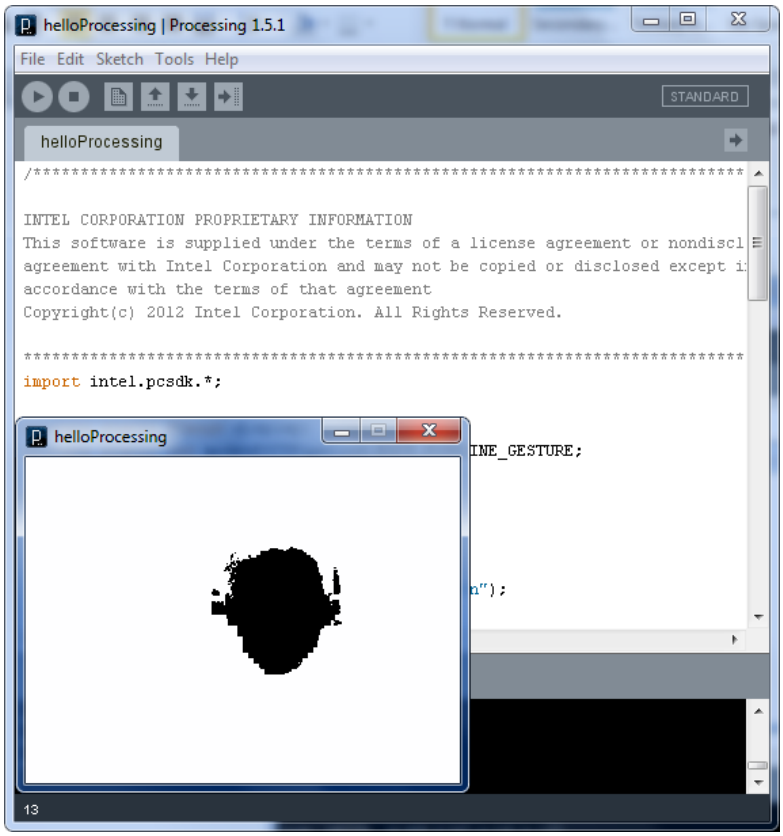

Figure 2: The Simple Processing* Application Screen Shot

openFrameworks Setup and Sample

The openFrameworks environment must use the following environment variables:

Include Path	`$(PCSDK_DIR)/include; $(PCSDK_DIR)/sample/common/include`
Link Path	`$(PCSDK_DIR)/lib/$(Platform);`
Link Library	`libpxc.lib`

Example 3 shows a simple openFrameworks application, which renders the label map image onto a texture. The screen shot is shown in Figure 3. The application uses the **`UtilPipeline`** utility class to create a finger tracking pipeline and retrieve the labelmap images.

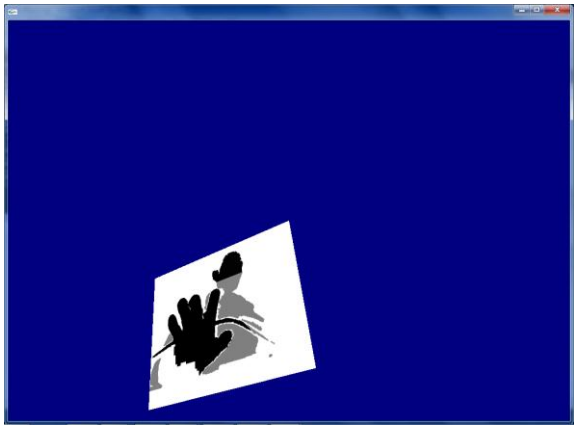

Figure 3: The Simple openFrameworks* Application Screen Shot

```
#include "ofAppGlutWindow.h"
#include "ofMain.h"
#include "util_pipeline.h"

class testApp:public ofBaseApp {
public:
    void setup();
    void draw();
protected:
    ofEasyCam      camera;
    ofTexture      texture;
    UtilPipeline   pp;
};

void testApp::setup() {
    ofSetVerticalSync(true);
    ofDisableNormalizedTexCoords();
    glEnable(GL_DEPTH_TEST);
    camera.setDistance(640);

    pp.EnableGesture();
    pp.Init();
    pxcU32 ww, hh;
    pp.QueryImageSize(PXCImage::IMAGE_TYPE_DEPTH,ww,hh);

    texture.allocate(ww,hh,GL_LUMINANCE);
    ofDisableDataPath();
}

void testApp::draw() {
    if (pp.AcquireFrame(false)) {
        PXCImage *im;
        if (pp.QueryGesture()->QueryBlobImage(PXCGesture::Blob::LABEL_SCENE,0,&im)
                                    >=PXC_STATUS_NO_ERROR) {
            PXCImage::ImageData data;
            if (im->AcquireAccess(PXCImage::ACCESS_READ,&data)>=PXC_STATUS_NO_ERROR) {
               texture.loadData(data.planes[0],texture.getWidth(),
                                    texture.getHeight(),GL_LUMINANCE);
               im->ReleaseAccess(&data);
            }
        }
        pp.ReleaseFrame();
    }
    ofBackground(0,0,128);
    ofSetColor(255);
    camera.begin();
        ofRotateX(ofRadToDeg(.5));
        ofRotateY(ofRadToDeg(-.5));
        texture.draw(0,0);
    camera.end();
}
int main() {
    ofAppGlutWindow window; // create a window
    ofSetupOpenGL(&window, 1024, 768, OF_WINDOW);
    ofRunApp(new testApp()); // start the app
}
```

Example 3: A Simple openFrameworks* Application

Interface and Function Reference

This section describes the SDK interfaces and their operations.

For the Unity game engine and the Processing frameworks, the member functions are under class **PXCUPipeline**. Use the following statements to initialize the class:

```
Unity:          PXCUPipeline pp=new PXCUPipeline();

Processing:     // this is an PApplet instance.
                PXCUPipeline pp=new PXCUPipeline(this);
```

AcquireFrame

Syntax

```
Unity:          bool AcquireFrame(bool wait);

Processing:     boolean AcquireFrame(boolean wait);
```

Parameters

wait	A boolean value to indicate if the function should block for processing results.

Description

This **AcquireFrame** function checks if any color, face, or gesture processing result is available. If the argument **wait** is true, the function blocks until some result is ready.

The function places a lock on the current processing result when ready. The application must call the **ReleaseFrame** function to continue processing the next frame.

The function returns **true** if the input device is disconnected. The application can use the **IsDisconencted** function to determine the device connection status.

Return Status

The function returns **true/false** to indicate the processing result is ready or not ready. The function returns false if there is some error and cannot continue processing.

Close

Syntax

```
Unity:          void Close(void);

Processing:     void Close(void);
```

Parameters

None

Description

This **`Close`** function destroys the processing pipeline and releases all resources.

Return Status

None

Init

Syntax

```
Unity:          bool Init(Mode mode);

Processing:     boolean Init(int mode);
```

Parameters

`mode`	The pipeline mode; see the **`Mode`** enumerator for definitions.

Description

This **`Init`** function initializes the color, face, or gesture processing pipeline. The mode parameter specifies the modules to be enabled in the pipeline.

Return Status

The function returns **`true/false`** to indicate the success or failure. For openFrameworks, the function returns the pipeline instance, or **`NULL`** if there is any failure.

IsDisconnected

Syntax

```
Unity:          bool IsDisconnected();

Processing:     boolean IsDisconnected();
```

Parameters

None

Description

This **IsDisconnected** function returns the device connection status.

Return Status

The function returns **true/false** to indicate if the device is disconnected.

PauseFrame

Syntax

```
Unity:          void PauseFrame(Mode mode, bool pause);

Processing:     void PauseFrame(int mode, boolean pause);
```

Parameters

mode	The pipeline mode; see the **Mode** enumerator for definitions.
pause	A boolean value to indicate pause or resume.

Description

This **PauseFrame** function pauses or resumes a specific pipeline module.

Return Status

None

QueryDepthMap

Syntax

```
Unity:          bool QueryDepthMap(short[] depthmap);

Processing:     boolean QueryDepthMap(short[] depthmap);
```

Parameters

`depthmap`	The depth map buffer

Description

This **`QueryDepthMap`** function returns the depth map data.

The depth map is a 2D array of 16-bit integers. Each pixel value indicates the distance from an object to the camera's XY plane or the Cartesian depth, in millimeters.

The application must pre-allocate the memory of the data buffer. The application can use the **`QueryDepthMapSize`** function to obtain the depth map resolution.

Return Status

The function returns **`true`** to indicate that the depth map is available, or **`false`** to indicate the depth map is not available or there is some internal error to retrieve the depth map data.

QueryDepthMapSize

Syntax

```
Unity:          bool QueryDepthMapSize(int[] size);

Processing:     boolean QueryDepthMapSize(int[] size);
```

Parameters

`size`	An integer array to return the resolution: **`size[0]`** is the width value and **`size[1]`** is the height value.

Description

This **`QueryDepthMapSize`** function returns the depth map resolution.

Return Status

The function returns **`true`** if the depth map is available, or **`false`** to indicate that the depth map is not available or there is some error to obtain the resolution.

QueryDeviceProperty

Syntax

Unity:	`bool QueryDeviceProperty(`**`PXCMCapture.Device.Property`** `pty, float[] data);`
Processing:	`boolean QueryDeviceProperty(int pty, float[] data);`

Parameters

`pty`	The device property identifier; see the **`PXCCapture::Device::Property`** enumerator for details.
`npty`	Number of properties to be retrieved
`data`	A floating-point array to retrieve the device properties

Description

This **`QueryDeviceProperty`** function returns the input device properties.

Return Status

The function returns **`true`** to indicate that the data is available, or **`false`** to indicate the data is not available.

QueryFaceID

Syntax

Unity:	`bool QueryFaceID(int fidx, out int face, out ulong timeStamp)`
Processing:	`boolean QueryFaceID(int fidx, long[] faceTimeStamp);`

Parameters

`fidx`	Zero-based index to retrieve all detected faces for the current frame.

`face`	Return the face identifier.
`timestamp`	Return the time stamp, in 100ns.
`faceTimeStamp`	Return the face identifier and time stamp, in 100ns, in a long integer array. The face identifier (an integer value) is in **`faceTimeStamp[0]`** and time stamp is in **`faceTimeStamp[1]`**.

Description

This **`QueryFaceId`** function returns the recognized face identifiers and time stamps for the current frame. The application needs to use the zero-based index to retrieve all available face identifiers.

Return Status

The function returns **`true`** if the data is available, or **`false`** if the data is not available, or there is any error to retrieve the data.

QueryFaceLandmarkData

Syntax

Unity:

```
bool QueryFaceLandmarkData(Int32 face,
PXCMFaceAnalysis.Landmark.Label label, int idx, out
PXCMFaceAnalysis.Landmark.LandmarkData data);
```

Processing:

```
boolean QueryFaceLandmarkData(int face, int label, int
idx, PXCMFaceAnalysis.Landmark.LandmarkData data);
```

Parameters

`face`	The face identifier, obtained from the **`QueryFaceID`** function.
`label`	The face landmark name. See the enumerator **`PXCFaceAnalysis::Landmark::Label`** for definitions.
`idx`	Zero-based index to enumerate all landmarks if the label parameter is a landmark group. For individual landmarks, the **`idx`** value must be zero.
`data`	The face landmark data defined in the **`PXCFaceAnalysis::Landmark::LandmarkData`** structure.

Description

This **QueryFaceLandmarkData** function returns individual face landmark data of the specified face identifier.

Return Status

The function returns **true** if the face landmark data is available, or **false** if such data is not available or there is some internal error to retrieve the data.

QueryFaceLandmarkData[]

Syntax

```
Unity:          bool QueryFaceLandmarkData(Int32 face,
                PXCMFaceAnalysis.Landmark.Label label,
                PXCMFaceAnalysis.Landmark.LandmarkData[] data);

Processing:     boolean QueryFaceLandmarkData(int face, int label,
                PXCMFaceAnalysis.Landmark.LandmarkData[] data);
```

Parameters

face	The face identifier, obtained from the **QueryFaceID** function.
label	The group face landmark name. See the enumerator **PXCFaceAnalysis::Landmark::Label** for definitions.
data	The array of face landmark data, in the **PXCFaceAnalysis::Landmark::LandmarkData** structure, to be returned.

Description

This **QueryFaceLandmarkData** function returns all face landmark data as defined in the landmark group for the specified face identifier.

Return Status

The function returns **true** if the face landmark data is available, or **false** if such data is not available or there is some internal error to retrieve the data.

QueryFaceLandmarkPose

Syntax

Unity: `bool QueryFaceLandmarkPose(Int32 face, out` **`PXCMFaceAnalysis.Landmark.PoseData`** `data);`

Processing: `boolean QueryFaceLandmarkPose(int face,` **`PXCMFaceAnalysis.Landmark_PoseData`** `data);`

Parameters

`face`	The face identifier, obtained from the **`QueryFaceID`** function.
`data`	The face landmark pose data defined in the **`PXCFaceAnalysis::Landmark::PoseData`** structure.

Description

This **`QueryFaceLandmarkPose`** function returns the face landmark pose data of the specified face identifier.

Return Status

The function returns **`true`** if the face landmark pose data is available, or **`false`** if such data is not available or there is some internal error to retrieve the data.

QueryFaceLocationData

Syntax

Unity: `bool QueryFaceLocationData(Int32 face, out` **`PXCMFaceAnalysis.Detection.Data`** `data);`

Processing: `boolean QueryFaceLocationData(int face,` **`PXCMFaceAnalysis.Detection_Data`** `data);`

Parameters

`face`	The face identifier, obtained from the **`QueryFaceID`** function.
`data`	The face location data defined in the **`PXCFaceAnalysis::Detection::Data`** structure.

Description

This **`QueryFaceLocationData`** function returns the face location data of the specified face identifier.

Return Status

The function returns **true** if the face location data is available, or **false** if such data is not available or there is some internal error to retrieve the data.

QueryGeoNode

Syntax

Unity:

```
bool QueryGeoNode(PXCMGesture.GeoNode.Label body, out PXCMGesture.GeoNode data);
bool QueryGeoNode(PXCMGesture.GeoNode.Label body, PXCMGesture.GeoNode[] darray);
```

Processing:

```
boolean QueryGeoNode(int body, PXCMGesture.GeoNode data);
boolean QueryGeoNode(int body, PXCMGesture.GeoNode[] darray);
```

Parameters

body	The geometric node identifier. See the **PXCGesture::GeoNode::Label** enumerator for definitions.
data	The geometric node data structure. See the **PXCGesture::GeoNode** structure definitions.
darray	An array of geometric node data structures. See the **PXCGesture::GeoNode** structure definitions.

Description

This **QueryGeoNode** function returns the details of a single node data structure or array of subsequent geometric node data.

Return Status

The function returns **true** if the geometric node data is available, or **false** if such data is not available or there is some internal error to retrieve the data.

QueryGesture

Syntax

Unity:

```
bool QueryGesture(PXCMGesture.GeoNode.Label body, out
```

```
                    PXCMGesture.Gesture data);

Processing:         boolean QueryGesture(int body, PXCMGesture::Gesture data);
```

Parameters

`body`	The body part that performs the pose/gesture. Use **`PXCMGesture.GeoNode.Label.LABEL_ANY`** (Unity), **`PXCMGsture.GeoNode.LABEL_ANY`** (Processing) for any available body part.
`data`	The pose/gesture details in the **`PXCGesture::Gesture`** structure.

Description

This **`QueryGesture`** function returns the current active pose/gesture details.

Return Status

The function returns **`true`** if the gesture data is available, or **`false`** if such data is not available or there is some internal error to retrieve the data.

QueryIRMap

Syntax

```
Unity:              bool QueryIRMap(short[] irmap);

Processing:         boolean QueryIRMap(short[] irmap);
```

Parameters

`irmap`	The IR or confidence map buffer

Description

This **`QueryIRMap`** function returns the IR or confidence map data.

The application must pre-allocate the memory of the data buffer. The application can use the **`QueryIRMapSize`** function to retrieve the IR or confidence map resolution.

IR or confidence map is a 2D array of 16-bit integers, indicating the confidence or intensity value of each depth pixel.

Return Status

The function returns **`true`** to indicate that the IR or confidence map is available, or **`false`** to indicate the IR or confidence map is not available or there is some internal error to

retrieve the IR or confidence map data.

QueryIRMapSize

Syntax

```
Unity:          bool QueryIRMapSize(int[] size);

Processing:     boolean QueryIRMapSize(int[] size);
```

Parameters

`size`	An integer array to return the resolution: **`size[0]`** is the width value and **`size[1]`** is the height value.

Description

This **`QueryIRMapSize`** function returns the IR or confidence map resolution.

Return Status

The function returns **`true`** if the IR or confidence map is available, or **`false`** to indicate that the IR or confidence map is not available or there is some error to obtain the resolution.

QueryLabelMap

Syntax

```
Unity:          bool QueryLabelMap(byte[] labelmap, int[] labels);

Processing:     boolean QueryLabelMap(byte[] labelmap, int[] labels);
```

Parameters

`labelmap`	The depth map data buffer.
`labels`	The buffer of labels.

Description

This **`QueryLabelMap`** function returns the label map data.

The application must pre-allocate the memory of the data buffer. The application can use

the **QueryLabelMapSize** function to retrieve the label map resolution.

Label map is an 8-bit gray image. Each pixel is labeled as background, left and right hands, etc. The pixel values are labeled in the **labels** integer array: **Labels[0]** is the background label, **labels[1]** is the left-hand label, and **labels[2]** is the right-hand label. If a certain label is not available, the label value is negative.

Return Status

The function returns **true** to indicate that the label map is available, or **false** to indicate the label map is not available or there is some internal error to retrieve the label map data.

QueryLabelMapAsImage

Syntax

```
Unity:          bool QueryLabelMapAsImage(Texture2D image);

Processing:     boolean QueryLabelMapAsImage(PImage image);
```

Parameters

image	The labelmap image. For Unity, the image is **Texture2D** and the color format is **ARGB32** and for Processing, the image is **PImage** and the color format is **RGB**.

Description

This **QueryLabelMap** function retrieves the label map as a native image.

The application must pre-allocate the image memory. The application can use the **QueryLabelMapSize** function to retrieve the label map resolution.

Return Status

The function returns **true** to indicate that the label map is available, or **false** to indicate the label map is not available or there is some internal error to retrieve the label map image.

QueryLabelMapSize

Syntax

```
Unity:          bool QueryLabelMapSize(int[] size);

Processing:     boolean QueryLabelMapSize(int[] size);
```

Parameters

`size`	An integer array to return the resolution: **`size[0]`** is the width value and **`size[1]`** is the height value.

Description

This **`QueryLabelMapSize`** function returns the label map picture resolution.

Return Status

The function returns **`true`** if the label map is available, or **`false`** to indicate that the label map is not available or there is some error to obtain the resolution.

QueryRGB

Syntax

```
Unity:          bool QueryRGB(Texture2D image);

Processing:     boolean QueryRGB(PImage image);
```

Parameters

`image`	The color picture image. For Unity, the image is **`Texture2D`** and the color format is **`ARGB32`**; for Processing, the image is **`PImage`** and the color format is **`RGB`**; and for openFrameworks, the image is an unsigned char buffer and the color format is **`RGB32`**.

Description

This **`QueryRGB`** function returns the color picture image.

The application must pre-allocate the memory of the image buffer. The application can use the **`QueryRGBSize`** function to query the resolution of the color picture.

Return Status

The function returns **`true`** to indicate that the color picture is available, or **`false`** to indicate the color picture is not available or there is some internal error.

QueryRGBSize

Syntax

```
Unity:          bool QueryRGBSize(int[] size);

Processing:     boolean QueryRGBSize(int[] size);
```

Parameters

`size`	An integer array to return the resolution: **`size[0]`** is the width value and **`size[1]`** is the height value.

Description

This **`QueryRGBSize`** function returns the color picture resolution, if available.

Return Status

The function returns **`true`** to indicate that the color picture is available, or **`false`** if the color picture is not available or there is some error to obtain the color picture resolution.

QueryUVMap

Syntax

```
Unity:          bool QueryUVMap(float[] uvmap);

Processing:     boolean QueryUVMap(float[] uvmap);
```

Parameters

`uvmap`	The UV map data buffer.

Description

This **`QueryUVMap`** function returns the UV map data.

The application must pre-allocate the memory of the UV map data buffer. The application can use the **`QueryUVMapSize`** function to obtain the size of the UV map data buffer.

The UV map is a 2D array. The UV map resolution is the same as the depth map resolution. Each UV map pixel contains two floating point values, which are the normalized **`XY`** coordinates in the color picture.

Return Status

The function returns **`true`** to indicate that the UV map is available, or **`false`** to indicate the UV map is not available or there is some internal error to retrieve the UV map data.

QueryUVMapSize

Syntax

```
Unity:          bool QueryUVMapSize(int[] size);

Processing:     boolean QueryUVMapSize(int[] size);
```

Parameters

`size`	An integer array to return the resolution: **`size[0]`** is the width value and **`size[1]`** is the height value.

Description

This **`QueryUVMapSize`** function returns the UV map resolution.

Return Status

The function returns **`true`** if the UV map is available, or **`false`** to indicate that the UV map is not available or there is some error to obtain the resolution.

QueryVoiceRecognized

Syntax

```
Unity:          bool QueryVoiceRecognized(out
                PXCMVoiceRecognition.Recognition data);

Processing:     boolean
                QueryVoiceRecognized(PXCMVoiceRecognition.Recognition
                data);
```

Parameters

`data`	Return the voice recognition data in the **`PXCMVoiceRecognition.Recognition`** structure.

Description

This **QueryVoiceRecognized** function returns any voice recognition data.

Return Status

The function returns **true** if there is voice recognition data available, or **false** if there is no data available.

ReleaseFrame

Syntax

```
Unity:          void ReleaseFrame(void);

Processing:     void ReleaseFrame(void);
```

Parameters

None

Description

This **ReleaseFrame** function releases the lock on any color, face, or gesture processing result and proceeds with processing the next frame.

Return Status

None

SetDeviceProperty

Syntax

```
Unity:          bool SetDeviceProperty(PXCMCapture.Device.Property pty,
                float[] data);

Processing:     boolean SetDeviceProperty(int pty, float[] data);
```

Parameters

pty	The device property identifier; see the **PXCCapture::Device::Property** enumerator for details.
data	A floating-point array to set the device properties

Description

This **`SetDeviceProperty`** function sets the device properties.

Return Status

The function returns **`true`** to indicate success, or **`false`** to indicate failure.

SetVoiceCommands

Syntax

```
Unity:          void SetVoiceCommands(String[] cmds);

Processing:     void SetVoiceCommands(String[] cmds);
```

Parameters

`cmds`	The voice command list.

Description

This **`SetVoiceCommands`** function sets the voice recognition module to be in the command and control mode. By default, the voice recognition module is in the dictation mode.

Return Status

None

SetVoiceDictation

Syntax

```
Unity:          void SetVoiceDictation();

Processing:     void SetVoiceDictation();
```

Parameters

`None`

Description

This `SetVoiceDictation` function sets the voice recognition module to be in the dictation mode. By default, the voice recognition module is in the dictation mode.

Return Status

None

Enumerator Reference

Mode

Description

The `Mode` enumerator uses bit-OR'ed values to select the components in the pipeline.

Name/Description

`COLOR_VGA`	Enable capturing VGA-resolution (640x480) color pictures
`COLOR_WXGA`	Enable capturing WXGA-resolution (1280x720) color pictures
`DEPTH_QVGA`	Enable capturing QVGA-resolution depth map pictures
`GESTURE`	Enable finger tracking and pose/gesture recognition
`FACE_LOCATION`	Enable face location detection/tracking
`FACE_LANDMARK`	Enable face feature-point detection/tracking

Name Space

`Processing`	The definitions are under name space **`PXCUPipeline`**.
`Unity`	The definitions are under name space **`PXCUPipeline.Mode`**.