INTEL® PERCEPTUAL COMPUTING SDK
Reference Manual
Core Framework

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.

OPENCL AND THE OPENCL LOGO ARE TRADEMARKS OF APPLE INC AND ARE USED BY PERMISSION BY KHRONOS.

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

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

Intel compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel Compiler User and Reference Guides” under "Compiler Options." Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20110307
# Table of Contents

Intel® Perceptual Computing SDK .......................................................... 1

Document Conventions ............................................................................. 1

Acronyms and Abbreviations ................................................................... 1

Architecture .............................................................................................. 2

Interface Hierarchy ................................................................................... 3

Simple Example ......................................................................................... 4

Programming Model .................................................................................. 5

SDK Interfaces ............................................................................................ 5

Algorithms and I/O Modules ....................................................................... 6

Module Configuration .................................................................................. 7

Build Processing Pipeline .......................................................................... 7

Data Passing and Asynchronous Execution .................................................. 8

Data Passing Using UtilCapture .................................................................. 12

Build Module Pipeline Using UtilPipeline .................................................. 13

Device Error Handling .............................................................................. 14

Image and Audio Data Abstraction .............................................................. 15

Map Depth Coordinates to Color Coordinates ............................................. 16

Color and Depth Picture Alignment ............................................................ 17

Interface and Function Reference .............................................................. 19

PXCAccelerator ............................................................................................ 19

CreateAudio ................................................................................................ 19

CreateImage ................................................................................................. 20
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryAccelType</td>
<td>21</td>
</tr>
<tr>
<td>QueryDevice</td>
<td>22</td>
</tr>
<tr>
<td>QueryHandle</td>
<td>22</td>
</tr>
<tr>
<td>SetDevice</td>
<td>22</td>
</tr>
<tr>
<td>SetHandle</td>
<td>23</td>
</tr>
<tr>
<td>PXCAudio</td>
<td>24</td>
</tr>
<tr>
<td>AcquireAccess</td>
<td>24</td>
</tr>
<tr>
<td>CopyData</td>
<td>25</td>
</tr>
<tr>
<td>QueryInfo</td>
<td>26</td>
</tr>
<tr>
<td>QueryOption</td>
<td>26</td>
</tr>
<tr>
<td>QueryTimeStamp</td>
<td>27</td>
</tr>
<tr>
<td>ReleaseAccess</td>
<td>27</td>
</tr>
<tr>
<td>SetOption</td>
<td>27</td>
</tr>
<tr>
<td>SetTimeStamp</td>
<td>28</td>
</tr>
<tr>
<td>TryAccess</td>
<td>28</td>
</tr>
<tr>
<td>PXCBase</td>
<td>29</td>
</tr>
<tr>
<td>DynamicCast</td>
<td>30</td>
</tr>
<tr>
<td>Release</td>
<td>30</td>
</tr>
<tr>
<td>PXCBaseImpl</td>
<td>31</td>
</tr>
<tr>
<td>PXCCallbackBase</td>
<td>31</td>
</tr>
<tr>
<td>DynamicCast</td>
<td>31</td>
</tr>
<tr>
<td>Release</td>
<td>32</td>
</tr>
<tr>
<td>PXCCapture</td>
<td>32</td>
</tr>
<tr>
<td>CreateDevice</td>
<td>33</td>
</tr>
<tr>
<td>QueryDevice</td>
<td>33</td>
</tr>
<tr>
<td>PXCCapture::AudioStream</td>
<td>34</td>
</tr>
</tbody>
</table>
QueryProfile ................................................................. 35
QueryStream ................................................................. 35
ReadStreamAsync .......................................................... 36
SetProfile ................................................................. 36
PXCCapture::Device ......................................................... 37
CreateStream ................................................................. 38
QueryDevice ................................................................. 39
QueryPropertyInfo .......................................................... 39
QueryProperty ................................................................. 40
QueryStream ................................................................. 41
SetPropertyAuto .............................................................. 41
SetProperty ................................................................. 42
PXCCapture::VideoStream ..................................................... 43
QueryProfile ................................................................. 43
QueryStream ................................................................. 44
ReadStreamAsync .......................................................... 44
SetProfile ................................................................. 45
PXCImage ................................................................. 46
AcquireAccess ............................................................... 46
CopyData ................................................................. 47
QueryInfo ................................................................. 48
QueryOption ................................................................. 48
QueryROI ................................................................. 49
QueryTimeStamp ............................................................ 49
ReleaseAccess .............................................................. 49
SetOption ................................................................. 50
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetROI</td>
<td>50</td>
</tr>
<tr>
<td>SetTimeStamp</td>
<td>51</td>
</tr>
<tr>
<td>TryAccess</td>
<td>51</td>
</tr>
<tr>
<td><strong>PXCMetadata</strong></td>
<td>52</td>
</tr>
<tr>
<td>AttachBuffer</td>
<td>53</td>
</tr>
<tr>
<td>AttachSerializable</td>
<td>53</td>
</tr>
<tr>
<td>CreateSerializable</td>
<td>54</td>
</tr>
<tr>
<td>DetachMetadata</td>
<td>55</td>
</tr>
<tr>
<td>QueryBuffer</td>
<td>55</td>
</tr>
<tr>
<td>QueryMetadata</td>
<td>56</td>
</tr>
<tr>
<td>QueryUID</td>
<td>56</td>
</tr>
<tr>
<td><strong>PXCPowerState</strong></td>
<td>57</td>
</tr>
<tr>
<td>QueryState</td>
<td>57</td>
</tr>
<tr>
<td>SetState</td>
<td>58</td>
</tr>
<tr>
<td><strong>PXCProjection</strong></td>
<td>58</td>
</tr>
<tr>
<td>MapColorToDepthCoordinates</td>
<td>59</td>
</tr>
<tr>
<td>MapDepthToColorCoordinates</td>
<td>60</td>
</tr>
<tr>
<td>ProjectImageToRealWorld</td>
<td>60</td>
</tr>
<tr>
<td>ProjectRealWorldToImage</td>
<td>61</td>
</tr>
<tr>
<td><strong>PXCScheduler</strong></td>
<td>62</td>
</tr>
<tr>
<td>QueryPriority</td>
<td>62</td>
</tr>
<tr>
<td>SetPriority</td>
<td>62</td>
</tr>
<tr>
<td><strong>PXCScheduler::SyncPoint</strong></td>
<td>63</td>
</tr>
<tr>
<td>QueryPtr</td>
<td>64</td>
</tr>
<tr>
<td>Synchronize</td>
<td>64</td>
</tr>
<tr>
<td>SynchronizeEx</td>
<td>65</td>
</tr>
<tr>
<td>Symbol</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
</tr>
<tr>
<td>PXCSession</td>
<td>66</td>
</tr>
<tr>
<td>CreateAccelerator</td>
<td>67</td>
</tr>
<tr>
<td>CreateImpl</td>
<td>67</td>
</tr>
<tr>
<td>CreateScheduler</td>
<td>69</td>
</tr>
<tr>
<td>LoadImplFromFile</td>
<td>69</td>
</tr>
<tr>
<td>QueryImpl</td>
<td>70</td>
</tr>
<tr>
<td>QueryVersion</td>
<td>70</td>
</tr>
<tr>
<td>UnloadImplFromFile</td>
<td>71</td>
</tr>
<tr>
<td>PXCSession_Create</td>
<td>71</td>
</tr>
<tr>
<td>PXCSmartArray</td>
<td>72</td>
</tr>
<tr>
<td>PXCSmartPtr</td>
<td>73</td>
</tr>
<tr>
<td>PXCSmartSP</td>
<td>74</td>
</tr>
<tr>
<td>PXCSmartSPArray</td>
<td>74</td>
</tr>
<tr>
<td>UtilCapture</td>
<td>75</td>
</tr>
<tr>
<td>LocateStreams</td>
<td>76</td>
</tr>
<tr>
<td>MapImages</td>
<td>77</td>
</tr>
<tr>
<td>QueryAudioStream</td>
<td>77</td>
</tr>
<tr>
<td>QueryCapture</td>
<td>78</td>
</tr>
<tr>
<td>QueryDevice</td>
<td>78</td>
</tr>
<tr>
<td>QueryImage</td>
<td>78</td>
</tr>
<tr>
<td>QueryVideoStream</td>
<td>79</td>
</tr>
<tr>
<td>ReadStreamAsync</td>
<td>79</td>
</tr>
<tr>
<td>SetFilter</td>
<td>80</td>
</tr>
<tr>
<td>UtilCaptureFile</td>
<td>80</td>
</tr>
</tbody>
</table>
QueryPosition ................................................................. 81
SetMask ........................................................................ 81
SetPause ....................................................................... 82
SetPosition .................................................................... 82
SetRealtime .................................................................. 82
UtilPipeline .................................................................... 83
AcquireFrame ................................................................. 85
Close ............................................................................. 86
EnableAudio .................................................................. 86
EnableFaceLandmark ....................................................... 87
EnableFaceLocation ......................................................... 88
EnableGesture ................................................................. 88
EnableImage .................................................................. 89
EnableVoiceRecognition .................................................. 89
Init ................................................................................ 90
IsAudioFrame ................................................................. 90
IsDisconnected ............................................................... 90
IsImageFrame ................................................................. 91
LoopFrames ................................................................... 91
OnAlert .......................................................................... 92
OnAudio ......................................................................... 92
OnFaceLandmarkSetup ..................................................... 93
OnFaceLocationSetup ....................................................... 93
OnFaceSetup .................................................................. 93
OnDisconnect ................................................................. 94
OnGesture ....................................................................... 94
PXCImage::SurfaceType ................................................................. 123
PXCPowerState::State ................................................................. 123
PXCScheduler::Priority .............................................................. 124
PXCSession::ImplGroup ............................................................. 125
PXCSession::ImplSubgroup ....................................................... 125
pxcStatus .................................................................................. 126
Appendices ................................................................................. 128
Appendix A: Data Type Definitions ........................................ 128
Appendix B: C# Language Support ............................................ 129
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 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 core framework of the Intel® Perceptual Computing SDK Application Programming Interface (API). The other SDK reference manuals that are released with the SDK describe different perceptual computing 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**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse-code modulation</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SP</td>
<td>Synchronization Point</td>
</tr>
</tbody>
</table>
Architecture

The SDK library architecture, as illustrated in Figure 1, consists of several layers of components. The essence of the SDK functionalities lies in the I/O modules and the algorithm modules. The I/O modules serve as sources that retrieve data from input devices or sinks that consume data to output devices. The algorithm modules include various pattern detection and recognition algorithms that are critical ingredients of innovative human computer experience, such as face recognition, gesture recognition, voice recognition, and text to speech.

The SDK standardizes the interfaces of the I/O modules and the algorithm modules so that the applications can access the functionalities without being concerned with the underlying implementations. Multiple implementations of SDK interfaces may coexist. The SDK core service provides the mechanism to search for a specific implementation from a set of available modules. The SDK core service also provides other critical features such as execution synchronization and interoperability with other libraries and frameworks.

On top of the standardized SDK interfaces, the SDK provides a set of utility classes for common usage cases. The SDK provides utility classes in source code form so that the developers can customize for their specific usages. The SDK also provides a few popular frameworks and language wrappers to extend the scope.

The applications are at the top of the library architecture. The applications link with and call into the SDK utilities, SDK interfaces, or SDK framework wrappers to perform operations. The SDK provides a rich set of samples to showcase the SDK functionalities as well as the SDK API usage.

![Figure 1: SDK Library Architecture Simplified](image)
Interface Hierarchy

Because the SDK targets different usage models as well as a wide variety of developers with different levels of expertise, it is difficult to design a single interface that works in all cases. Some developers value simplicity and just want to get the job done. Others look for controls that can fine-tune every step of the algorithms. The SDK provides layers of the interface hierarchies to address these needs.

As illustrated in Figure 2, the basic SDK building blocks are a set of C++ interfaces, defining the functionalities of core frameworks, I/O modules, and algorithm modules. These interfaces provide the raw functionalities. For example, the PXCCapture interface only retrieves audio samples (PXCAudio) or image samples (PXCImage) from input sensors. The PXCCapture interface performs finger tracking and gesture recognition with the right type of input image samples. The applications using these interface levels achieve the maximum flexibilities possible with this SDK, with moderate complexity in programming. There is a straight port of all the C++ interfaces to C# on the left side.

The SDK provides a set of utility classes to simplify application programming. The UtilCapture class is an extension of the PXCCapture interface to help connect algorithm modules with the input devices and stream data between them. The UtilPipeline class provides a simple interface if the usage is limited to gesture, face, and voice algorithms. With the exception of the openFrameworks*, the SDK bases its framework ports on the PXCPipeline pipeline, a customization of the UtilPipeline class in C and for framework support (for example, converting images to framework native formats.)

Your application is free to use any of these interfaces or customize them for special usages. For utmost simplicity, you may want to start with the UtilPipeline class. For more flexibility, you can use UtilCapture and the SDK building block interfaces. Most applications will need to use certain combinations. For example, let the UtilPipeline class figure out how to stream data from the input device to the algorithm modules, and use the PXCGesture interface to control the algorithm details.

![Figure 2: SDK Interface Hierarchy](image-url)
Simple Example

Example 1 shows an application that uses the `UtilPipeline` class for gesture recognition. The application enables gesture recognition by using the `EnableGesture` function. It enters the streaming loop by using the `LoopFrames` function, which initializes the gesture recognition pipeline and streams data from the input device, a depth camera, to the gesture recognition module.

```cpp
#include "util_pipeline.h"

class MyPipeline: public UtilPipeline {
    public:
        MyPipeline(void):UtilPipeline() {
            EnableGesture();
            nframes=0;
        }
        virtual void PXCAPI OnGesture(PXCGesture::Gesture *data) {
            wprintf(L"OnGesture(label=%d)\n", data->label);
        }
        virtual bool OnNewFrame() {
            return (nframes++<5000);
        }
    protected:
        int nframes;
};

int wmain(int argc, WCHAR* argv[]) {
    MyPipeline pipeline;
    pipeline.LoopFrames();
    return 0;
}
```

Example 1: A Simple Gesture Recognition Application

On each delivered frame of data, the SDK invokes the `OnNewFrame` function for customized operations. The return value also signals if the streaming loop should terminate. In this example, the application streams 5000 frames and then terminates. When a gesture is recognized, the SDK invokes the `OnGesture` function, with the gesture details as input arguments.

The `UtilPipeline` utility class is what most applications can use that need only a simple pipeline for gesture recognition and face analysis. For more complicated usages, the SDK provides additional layers of functions. The remaining sections cover the basics of SDK programming so application developers can have a global view of how things are done, as well as examples of how higher level utility classes may simplify programming.
Programming Model

The SDK general programming procedure is as follows, as illustrated in Example 2:

1. **Session Creation:** An SDK session is the very first object any SDK application must create. “Session” is a context that the SDK uses to hold all I/O or algorithm modules. The application can use session functions to create a module instance. The application calls the `PXCSession_Create` function to create an SDK session.

2. **Module Creation:** The application creates an instance of some algorithm, I/O module, or service by calling the `CreateImpl` function, optionally with the help of the `QueryImpl` function to enumerate available algorithms or I/O modules.

3. **Module Operation:** The application calls the member functions of the module instance for certain functionalities.

4. **Close Down:** Finally, the application must release all created instances.

```c++
// Create a session instance
PXCSmartPtr<PXCSession> session;
PXCSession_Create(&session);

// Create an algorithm instance
PXCSmartPtr<PXCCapture> capture;
session->CreateImpl(PXCCapture::CUID, (void**)&capture);
```

**Example 2: SDK Application Programming Procedure**

Note that in Example 2, the application uses the `PXCSmartPtr` template functions to manage the life cycle of the `session` and `capture` instances, which simplifies application programming. A few of these templates, such as `PXCSmartArray`, `PXCSmartSP`, and `PXCSmartSPArray`, are for convenience. Not a must.

SDK Interfaces

SDK interfaces (those with the `PXC` prefix) are defined as C++ virtual classes, each of which may have multiple implementations. Applications must observe restriction and that is the application should not use the compiler keyword `dynamic_cast` to cast the types. Instead, use the `DynamicCast` template to query a specific interface from the module implementation, as illustrated in Example 3.

```c++
PXCFaceAnalysis::Detection* ToDetection(PXCFaceAnalysis *face) {
    return face->DynamicCast<PXCFaceAnalysis::Detection>();
}
```

**Example 3: Dynamic Cast Between SDK Interfaces**
Note that utility classes, such as `UtilCapture` and `UtilPipeline` are regular C++ classes thus the above restriction does not apply to them.

## Algorithms and I/O Modules

An SDK session may contain multiple algorithm and I/O module implementations. At the session creation time, the SDK loads all preinstalled modules into the session. The application may also explicitly load a module by calling the `LoadImplFromFile` function, as illustrated by Example 4.

```cpp
// The application explicitly load a module
session->LoadImplFromFile(L"my_module.dll");
```

**Example 4: Explicitly Load Modules Into A Session**

The application must create an instance of a module before using its functions. There are multiple ways that the application can locate a module and create an instance.

The simplest way of creating a module instance is to create a module instance based on its interface. As illustrated by Example 2, the application creates an instance of the capture module by using the `CreateImpl` function with the interface identifier `PXCCapture::CUID`. The SDK searches for the first module that implements the specified interface and creates an instance of the module. This works best if the application simply uses the interface functionalities and does not customize the results for a particular module.

In some cases, the application may need to pinpoint a module for customization. This can be done in two ways:

1. If the application already knows the module’s implementation identifier, the application can call the `CreateImpl` function with the implementation identifier, as illustrated in Example 5. The interface identifier and the implementation identifier uniquely identify a module implementation. The application can find the implementation identifier by calling the `QueryImpl` function, which enumerates all available modules.

```cpp
// Create an algorithm instance
PXCFaceAnalysis *fd=0;
session->CreateImpl(PXC_UID('F','I','L','C'),
PXCFaceAnalysis::CUID,(void**)&fd);
```

**Example 5: Create Module Instance with Implementation Unique Identifier**

2. The application can also fill out an `ImplDesc` structure that details different aspects of a module. The `CreateImpl` function will find a match and create an instance of the module, as illustrated by Example 6. Here the `ImplDesc` structure serves as a search template if not
completely filled. Zero values in a structure field will match any values. A bit-pattern in a bit-OR'ed field will match any bit patterns that are a superset.

```c
PXCSession::ImplDesc desc;
memset(&desc,0,sizeof(desc));
desc.group=PXCSession::IMPL_GROUP_OBJECT_RECOGNITION;
desc.subgroup=PXCSession::IMPL_SUBGROUP_FACE_ANALYSIS;
desc.vendor=0x8086; // from Intel 
// Create an algorithm instance
PXCFaceAnalysis *fd=0;
session->CreateImpl(&desc,PXCFaceAnalysis::CUID,(void**)&fd);
```

Example 6: Create Module Instance with Implementation Descriptor

### Module Configuration

It is important to configure a module implementation after creating its instance, as the module may support different configurations. Each module interface usually exposes two configuration member functions: `QueryProfile` and `SetProfile`. The former function enumerates all supported configurations or queries the current working configuration, while the latter function sets the active configuration, as illustrated in Example 7.

```c
PXCFaceAnalysis::ProfileInfo info;
// enumerate the first supported configuration
sts=face->QueryProfile(i,&info);
// set the working configuration
face->SetProfile(&info);
...
// obtain the current working configuration
face->QueryProfile(&info);
```

Example 7: Working with Module Configurations

By default, a module is not initialized after creation, the `SetProfile` function also serves as a means to initialize the module. After the initial setup, the application can also call the `SetProfile` function to change some working parameters during runtime.

### Build Processing Pipeline

Algorithm modules usually do not work by themselves. The application needs to identify a source to provide input data to them. If the application has prior knowledge of the algorithm module and its input device, the application can directly create a source device module and a data processing module, and pass data between the two modules, as illustrated in Example 8.
To write generic code, the application may want to retrieve the input data needed from the algorithm configuration, and then search for a qualifying input device. The `UtilCapture` utility class does just that. The `LocateStreams` function enumerates available source devices and matches the source to the algorithm configurations, as illustrated in Example 8.

Note that after the `LocateStreams` function call, the `inputs` data structure that specifies the algorithm module input needs is altered with the current input device properties. It is critical that the application sets the algorithm module configuration after the `LocateStreams` function so that the current input device properties can be passed to the algorithm module.

Besides matching a single algorithm module to an input device, the `LocateStreams` function can consolidate multiple input requirements and find a qualifying input device, as illustrated in Example 9, where the function locates a device that can provide data to both the face and gesture modules.

```
// retrieve the input data needs from a face analysis module
PXCFaceAnalysis::ProfileInfo info;
face->QueryProfile(0, &info);

// match a capturing device that can provide qualifying data
UtilCapture capture(session);
capture.LocateStreams(&info.inputs);

// Set the module configuration
face->SetProfile(&info);
```

Example 8: `UtilCapture` to Find a Match to a Capture Source Device

```
PXCFaceAnalysis::ProfileInfo pinfo1;
face->QueryProfile(0, &pinfo1);

PXCGesture::ProfileInfo pinfo2;
gesture->QueryProfile(0, &pinfo2);

UtilCapture capture(session);

std::vector<PXCCapture::VideoStream::DataDesc*> minputs;
minputs.push_back(&pinfo1.inputs);
minputs.push_back(&pinfo2.inputs);
capture.LocateStreams(minputs);

face->SetProfile(&pinfo1);
gesture->SetProfile(&pinfo2);
```

Example 9: `UtilCapture` to Find a Match to Multiple Input Needs

**Data Passing and Asynchronous Execution**

The SDK design requires an application to explicitly pass data from modules to modules for the required functionalities. Such data passing and function execution occur at runtime and form an
operation pipeline. For example, to detect face location, as illustrated in Figure 3, the application creates a run-time pipeline consisting of camera input and face detection and explicitly pass the samples from the camera input to the face detection module.

![Camera Input](image1)

**Figure 3: Face Detection Pipeline**

To improve execution efficiency and avoid any threading-related complexity, the SDK designs each stage of pipeline operation to be asynchronous. In Figure 3, the application passes the data from the camera input to the face detection module asynchronously, without waiting for the input samples to be ready, or synchronizing between the two modules.

The SDK marks asynchronous functions with a suffix “Async” in the function name. Asynchronous functions do not block execution and return a synchronization point (abbreviated as SP) for later synchronization of the execution result. The application uses the SP member function **Synchronize** to explicitly wait for execution completion. Each asynchronous operation returns a SP. The application can ignore the SP1 synchronizations (recommended) and only synchronize at the end of the pipeline, SP2. In this case, the SDK maintains the SP1 synchronization automatically.

![Camera Input](image2)

**Figure 4: Face Detection Asynchronous Pipeline with SP**

Unless explicitly stated in the function description, the application needs to consider any output data from an upstream operation as unavailable. The output data becomes available after the application synchronizes the corresponding SP or the end SP of the corresponding asynchronous pipeline. In Figure 5, the image data is not available at the time the application invokes the face detection function. The application simply passes the image buffer pointer from the camera input function to the face detection function. The SDK will fill the image buffer content at a later time when a sample is available from the camera. Reasonably, the application should not
deallocation any buffers or structures before the application synchronizes the entire pipeline execution, as their life cycles are dynamic during execution.

The SDK supports asynchronous execution as a graph. There are two ways to construct an execution graph:

- The application passes the same output to different subsequent executions.
- Some SDK functions generate multiple outputs. The application passes each output to a different subsequent execution.

![Figure 6: Asynchronous Execution Graph](image)

In Figure 6, function 1 generates two outputs. The application passes the first output through function 2 and function 4, and the second output through function 3 and function 5. The application also passes the function 2 output to function 5 as function 5 requires multiple inputs. During such an asynchronous graph execution, the following rules apply:

- For SDK functions that generate multiple outputs, the SDK considers each output an individual entity in terms of activating or aborting subsequent executions. For example, if the function 1 output 1 is ready, the SDK will invoke function 2 execution while function 1 output 2's execution is in progress. If the function 1 output 1 is aborted but function 1 output 2 can proceed, the SDK will abort function 2 and 4 executions while still proceeding with function 3 and function 5 executions.
- For SDK functions that require multiple inputs, the SDK activates the function execution only when all inputs are ready from upstream executions.
- When the application synchronizes an SP, the SDK waits until all function outputs are ready. It is not possible to wait for a single output. However, the application may use the SP member function `QueryPtr` to check each output execution status.
- The application may use the SP member function `SynchronizeEx` to synchronize on multiple SPs.
- The application may delete any intermediate SP after connecting all dependency functions if the application does not need them for error recovery. For example, the application may delete SP1 (of function 1) after connecting function 2 and function 3 to function 1.

Asynchronous pipeline execution is critical to achieve the best performance. The more functions that can be asynchronously connected, the better the underlying pipelining. It is highly recommended that the application does not synchronize immediately after calling an
asynchronous function, which effectively stalls the execution. For example, in Example 10, each stream is read and rendered sequentially. Stream 1 operations block stream 2 operations, and vice versa. The same task can be rewritten as in Example 11. Each stream is read and rendered asynchronously thus stream 1 and stream 2 operations can be in parallel.

```
while (!stop) {
    for (i=0;i<2;i++) {
        PXCSmartPtr<PXCImage> image;
        PXCSmartSP sp;
        stream[i]->ReadStreamAsync(&image, &sp);
        sp->Synchronize();
        // assume rendering is a blocking call.
        renders[i]->RenderFrame(image);
    }
}
```

**Example 10: Render Two Streams Synchronously (Not Recommended)**

```
PXCSmartArray<PXCImage> images(2);
PXCSmartSPArray sp(2);

// initialization
for (i=0;i<2;i++)
    streams[i]->ReadStreamAsync(&images[i],&sp[i]);

while (!stop) {
    // which stream is ready?
    sp->SynchronizeEx(&i);
    // render the stream that is ready.
    renders[i]->RenderFrame(image);
    // read another sample
    streams[i]->ReadStreamAsync(images.ReleaseRef(i),sp.ReleaseRef(i));
}

// wait for any remaining reads
sp->SynchronizeEx();
```

**Example 11: Render Two Streams Asynchronously (Recommended)**

Errors may occur at any stage of an asynchronous pipeline execution. The SDK uses the scheme shown in Table 1 to report the errors:

- If the error occurs in the stage where the application synchronizes the SP, the SDK returns the corresponding error code.
- Otherwise, the SDK returns `PXC_STATUS_EXECUTION_ABORTED`. The application needs to query any upstream SP for the exact error location and status code.

<table>
<thead>
<tr>
<th>Error Occurrence</th>
<th>SP Status</th>
<th>SP1</th>
<th>SP2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 1: Error Reporting Scheme in Asynchronous Pipeline Execution

<table>
<thead>
<tr>
<th>Face Detection</th>
<th>NoError*</th>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera Input</td>
<td>Error</td>
<td>ABORTED**</td>
</tr>
</tbody>
</table>

* PXC_STATUS_NO_ERROR
** PXC_STATUS_EXECUTION_ABORTED

Data Passing Using UtilCapture

As mentioned in the Build Processing Pipeline section, the UtilCapture utility class can help identify an input device that matches a single or multiple algorithm modules.

It is relatively straight forward to pass data from the input device to a single algorithm module. The application calls theReadStreamAsync function to retrieve the sample from the input device, and then passes it to the face detection module as illustrated in Example 12.

```
PXCSmartSPArray sps(2);
PXCSmartArray<PXCImage> images;
capture.ReadStreamAsync(images,&sps[0]);
face->ProcessImageAsync(images,face,&sps[1]);
while (...) {
    /* do something here */
    sps.SynchronizeEx();
    capture.ReadStreamAsync(images.ReleaseRefs(),sps.ReleaseRef(0));
    face->ProcessImageAsync(images,face,sps.ReleaseRef(1));
}
```

Example 12: UtilCapture to Stream a Simple Pipeline

When passing the samples from the input device to multiple modules, the application needs to map the view of the input samples. As illustrated in Figure 7, the input requirements of processing modules are specified as DataDesc1 and DataDesc2, respectively. For example, module 1 may need a color stream, and module 2 may need a color stream and a depth stream. The LocateStreams function of the UtilCapture interface consolidates these requests and locates an input device that can provide data for both modules.

During streaming, the input device (through the ReadStreamAsync function) generates a set of samples, described as Images. The order of samples is based on the input device definition (device view). The MapImages function maps the samples from the device view to the module view, Images1 (color only) and Images2 (color and depth, in that order). The application then feeds the data to the processing modules, as illustrated in Example 13.
**Figure 7: Multiple Processing Modules Pipeline**

```cpp
PXCSmartSPArray sps(3);
PXCSmartArray<PXCImage> images;
PXCCapture::VideoStream::Images images1, images2;

capture.ReadStreamAsync(images,&sps[0]);
capture.MapImages(images, images1);
face->ProcessImageAsync(images1,&sps[1]);
capture.MapImages(images, images2);
gesture->ProcessImageAsync(images2,&sps[2]);
while (...) {
  /* do something here */
  sps.SynchronizeEx();
  capture.ReadStreamAsync(images.ReleaseRefs(),sps.ReleaseRef(0));
capture.MapImages(images, images1);
face->ProcessImageAsync(images1,sps.ReleaseRef(1));
capture.MapImages(images, images2);
gesture->ProcessImageAsync(images2,sps.ReleaseRef(2));
}
```

**Example 13: UtilCapture to Stream A Complex Pipeline**

**Build Module Pipeline Using UtilPipeline**

The SDK provides maximum flexibility by exposing the details of pipeline building and data passing among pipeline components. For simpler usages, the application should use the **UtilPipeline** utility class, which hides the pipeline complexity.

**Figure 8: UtilPipeline Pipeline Model**
As illustrated in Figure 8, the **UtilPipeline** utility implements a pipeline from either a live input device, or file-based playback. The utility hard-codes the algorithm modules to be the face detection/landmark detection module and the gesture recognition module.

The application usually derives a class from the **UtilPipeline** utility class for overriding certain callback functions. The general procedure of using the **UtilPipeline** utility is as follows:

- **Configuration**: The application uses the `EnableXXXX` set of functions to configure the pipeline. For example, use the `EnableGesture` function to enable the gesture module in the pipeline, or use the `EnableFileMode` function to setup recording.
- **Run the pipeline**: The application calls the `LoopFrames` function to initialize the pipeline and pass data among the pipeline components.
- **Handle events**: The application overrides the `OnXXXX` function to handle events from pipeline event notification. Particularly, the return value of the `OnNewFrame` function signals when to exit the pipeline.

See the Simple Example section for an example and the **UtilPipeline** detailed definitions.

### Device Error Handling

"Device" refers to any peripheral device that an SDK I/O module operates on. If during operation, a device generates an error, the SDK returns a status code, as listed in Table 2. The application must handle them properly.

<table>
<thead>
<tr>
<th>Error Status</th>
<th>Description</th>
<th>Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PXC_STATUS_DEVICE_FAIL</code></td>
<td>Unexpected device failure due to device malfunctioning.</td>
<td>The application must close the module instance and recreate it.</td>
</tr>
<tr>
<td><code>PXC_STATUS_DEVICE_LOST</code></td>
<td>Unexpected device error due to the loss of the device such as system sleep/hibernation, or unplug of the peripheral device.</td>
<td>The application can choose to abort or wait for the device to reconnect. In the latter case, the device operation will resume.</td>
</tr>
<tr>
<td><code>PXC_STATUS_DEVICE_BUSY</code></td>
<td>The application does not have control of the device thus cannot change the device configuration.</td>
<td>The application can only use the existing configuration of the device.</td>
</tr>
</tbody>
</table>

*Table 2: Device-related Error Status Codes*

If an I/O module supports device disconnection and reconnection, the I/O module behaves as follows:
Upon device disconnection, the I/O module returns \texttt{PXC\_STATUS\_DEVICE\_LOST} (as a comparison, if a module does not support device disconnection, the return code is instead \texttt{PXC\_STATUS\_DEVICE\_FAILED}). The error code can be directly from the module's \texttt{ReadStreamAsync} function return code, or from the SP synchronization result.

- If the application chooses to abort, the application can destroy all allocated resources and exit.
- If the application chooses to wait for device reconnection, the application can periodically try to stream data. The I/O module will return \texttt{PXC\_STATUS\_NO\_ERROR} instead of \texttt{PXC\_STATUS\_DEVICE\_LOST} if the device is reconnected.

Example 14 shows application code to handle device disconnection and reconnection.

```c
for (;;) {
    sts=captureReadStreamAsync(images.ReleaseRefs(),sps.ReleaseRef(0));
    if (sts==PXC\_STATUS\_DEVICE\_LOST) { Sleep(5); continue; }
    if (sts<PXC\_STATUS\_NO\_ERROR) break; // EOF

    gestureDetector-\gt ProcessImageAsync(images,sps.ReleaseRef(1));
    sps.SynchronizeEx();
    sts=sps[0]-\gt Synchronize(0);
    if (sts==PXC\_STATUS\_DEVICE\_LOST) { Sleep(5); continue; }
    if (sts<PXC\_STATUS\_NO\_ERROR) break; // EOF

    // Process gesture recognition results.
}
```

\textbf{Example 14: Handle Device Disconnect/Reconnect}

\section*{Image and Audio Data Abstraction}

The SDK uses the \texttt{PXCImage} interface to abstract images and the \texttt{PXCAudio} interface to abstract audios, for interoperability with other libraries or any OS-specific surfaces, such as Microsoft DirectX* surfaces.

Each image or audio object is associated with a storage type, and the \texttt{PXCAccelerator} instance maintains the context of each storage type. For example, the system memory storage can create image or audio samples in the system memory buffer, and the Microsoft DirectX 9 storage can create image and audio samples in the DirectX 9 surfaces. To create image or audio objects, the application first needs to create the storage object (\texttt{CreateAccelerator}), then create the corresponding image (\texttt{CreateImage}) and audio (\texttt{CreateAudio}) objects.

To access an image buffer, the application uses the \texttt{AcquireAccess} function to request exclusive access to the image data. The function returns the internal data buffer details. After reading and/or writing to the image data, the application uses the \texttt{ReleaseAccess} function to revoke the exclusive access. Alternatively, the application can use the \texttt{CopyData} function to
copy an image from a source `PXCImage` object. The same procedure applies to the audio buffer creation and access. The application must use the corresponding set of audio functions. See the `PXCAudio` class member functions for details.

When done with the image or audio objects, the application must release the image or audio instances.

### Map Depth Coordinates to Color Coordinates

Due to cameras’ physical location, lens size, and field of view differences, depth pixel coordinates do not map 1:1 to color pixel coordinates. The UV map provides a full mapping of every depth pixel coordinate to the color pixel coordinate.

The UV map is part of all depth image samples, which consist of three planes: (a) Plane 0: depth map or vertices; (b) Plane 1: confidence map, and (c) Plane 2: uvmap. The resolution of the UV map is the same as the depth map. Each UV map pixel contains two floating point numbers, or normalized coordinates (0–1), in the color picture resolution.

The application can map the depth coordinates to the color coordinates as shown in Example 15.

```cpp
void MapXY(float &x, float &y, PXCImage* depthSample, PXCImage::ImageInfo *colorInfo) {  
PXCImage::ImageData ddata;  
depthSample->AcquireAccess(PXCImage::ACCESS_READ,&ddata);  
float *uvmap=(float*)ddata.planes[2];  
int index=((int)y)*depthInfo->width+x;  
x=uvmap[index*2]*colorInfo->width;  
y=uvmap[index*2+1]*colorInfo->height;  
depthSample->ReleaseAccess(&ddata); }
```

**Example 15: Map Depth Coordinates to Color Coordinates**

Alternatively, some camera provides the `PXCProjection` interface for explicit conversion among 2D depth coordinates, 2D color coordinates, and 3D real world coordinates. The I/O module serializes the `PXCProjection` implementation into the session metadata and provides the metadata identifier through device property `PROPERTY_PROJECTION_SERIALIZABLE`. Example 16 shows how to map depth coordinates to color coordinates through the `PXCProjection` interface.
Color and Depth Picture Alignment

The SDK capture module exposes color and depth streams as individual streams. Normally, the application reads each stream independently. In some cases, the application may want to align the depth stream to the color stream. This can be done by synchronizing the read operation of both streams.

The `ReadStreamAsync` function in the `UtilCapture` utility class is designed to perform synchronized reads of multiple streams. As illustrated in Example 17, the application specifies the color and depth pictures as the input requirement, and then calls the `ReadStreamAsync` function in a loop. Once the SP is synchronized, the color and depth samples are ready for processing.

Alternatively, as shown in Example 18, developers can use the `UtilPipeline` functions to achieve synchronized reads. The application calls the `AcquireFrame(true)` function to block the execution until both color and depth samples are ready.

```c
void MapXY(float &x, float &y, float d, PXCCapture::Device *dev, PXCSession *ss) {
    pxcUID uid;
    dev->QueryPropertyAsUID(PXCCapture::Device::PROPERTY_PROJECTION_SERIALizable, &uid);
    PXCSmartPtr<PXCProjection> pj;
    session->DynamicCast<PXCMetadata>()->CreateSerializable<PXCProjection>(uid, &pj);
    PXCPose3DF32 posd = { x, y, d };
    PXCPoseF32 posc;
    pj->MapDepthToColorCoordinates(1, &posd, &posc);
    x = posc.x; y = posc.y;
}
```

Example 16: Map Depth Coordinates to Color Coordinates via the PXCProjection Interface
UtilCapture capture(session);
PxCCapture::VideoStream::DataDesc request;
memset(&request, 0, sizeof(request));
request.streams[0].format=PXCImage::COLOR_FORMAT_RGB32;
request.streams[1].format=PXCImage::COLOR_FORMAT_DEPTH;
capture.LocateStreams (&request);

for (;;) {
    PXCSmartArray<PXCImage> images(2);
    PXCSmartSP sp;
    
captureReadStreamAsync(images, &sp);
    sp->Synchronize();
    // Color picture in images[0] and depth picture in images[1]
    // Process color and depth images
}

Example 17: Aligned Color & Depth Reads using UtilCapture

UtilPipeline mypipeline;
mypipeline.EnableImage(PXCImage::COLOR_FORMAT_RGB32);
mypipeline.EnableImage(PXCImage::COLOR_FORMAT_DEPTH);
for (;;) {
    mypipeline.AcquireFrame(true);
    PXCImage *rgb=mypipeline.QueryImage(PXCImage::IMAGE_TYPE_COLOR);
PXCImage *depth=mypipeline.QueryImage(PXCImage::IMAGE_TYPE_DEPTH);
mypipeline.ReleaseFrame();
}
mypipeline.Close();

Example 18: Aligned Color & Depth Reads using UtilPipeline
Interface and Function Reference

This section describes SDK interfaces and their operations.

In each interface description, only commonly used status codes are documented. The function may return additional status codes in certain cases. See the `pxcStatus` enumerator for a list of all status codes.

**PXCAccelerator**

The `PXCAccelerator` interface manages the device context of image and audio storage. Each `PXCAccelerator` instance manages a specific device context. Among the supported acceleration frameworks, the CPU accelerator (of acceleration type `ACCEL_TYPE_CPU`) is the default accelerator in SDK sessions. The CPU accelerator is also a singleton object. Multiple calls to create the accelerator will return the same instance.

The application can use the `CreateAccelerator` function to create an instance of the `PXCAccelerator` interface.

The `PXCAccelerator` interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>QueryAccelType</code></td>
<td>Query the hardware acceleration framework</td>
</tr>
<tr>
<td><code>SetDevice</code></td>
<td>Set the hardware acceleration device</td>
</tr>
<tr>
<td><code>QueryDevice</code></td>
<td>Query the hardware acceleration device</td>
</tr>
<tr>
<td><code>SetHandle</code></td>
<td>Set the OS-specific handle for hardware acceleration</td>
</tr>
<tr>
<td><code>QueryHandle</code></td>
<td>Query the OS-specific handle</td>
</tr>
<tr>
<td><code>CreateImage</code></td>
<td>Create an instance of the <code>PXCImage</code> interface</td>
</tr>
<tr>
<td><code>CreateAudio</code></td>
<td>Create an instance of the <code>PXCAudio</code> interface</td>
</tr>
</tbody>
</table>

**CreateAudio**

**Syntax**

```cpp
pxcStatus CreateAudio(PXCAudio::AudioInfo *info, PXCAudio::AudioOption)
```
options, PXCAudio::AudioData *data, PXCAudio **audio);
pxcStatus CreateAudio(PXCAudio::AudioInfo *info, PXCAudio **audio);

Parameters

info
The AudioInfo structure for audio buffer information.

options
Audio creation options. See the AudioOption enumerator for details.

data
Optional initialization data; see the AudioData structure for details.

audio
The PXCAudio instance to be returned.

Description

This CreateAudio function creates an instance of the PXCAudio interface, which manages audio buffer access.

If data!=0, the SDK initializes the audio buffer with the specified data, by taking the buffer pointers or the OS-specific surface handle, without any data copying. The format in the AudioData structure must be the same as in the AudioInfo structure.

If data=0, the SDK creates the audio buffer without initialization.

Return Status

PXC_STATUS_NO_ERROR The function returned successfully.

PXC_STATUS_ALLOC_FAILED Failed to allocate the audio buffer.

Change History

This function was introduced in SDK API 1.0.

CreateImage

Syntax

pxcStatus CreateImage(PXCImage::ImageInfo *info, PXCImage::ImageOption options, PXCImage::ImageData *data, PXCImage **image);
pxcStatus CreateImage(PXCImage::ImageInfo *info, PXCImage **image);

Parameters

info
The required image properties in the ImageInfo structure.
options  
Image creation options. See the ImageOption enumerator for details.

data  
Optional initialization data; see the ImageData structure for details.

image  
The instance of the PXCImage interface to be returned.

Description

This function creates an instance of the PXCImage interface to manage image buffer access.

If data!=0, the SDK initializes the image buffer with the specified data, by taking the buffer pointers or the OS-specific surface handle, without any data copying. The formats in the ImageInfo structure and in the ImageData structure must match.

If data==0, the SDK creates the image buffer without initialization.

Return Status

PXC_STATUS_NO_ERROR  
The function returned successfully.

PXC_STATUS_ALLOC_FAILED  
Failed to allocate the image buffer.

Change History

This function was introduced in SDK API 1.0.

---

**QueryAccelType**

Syntax

AccelType QueryAccelType(void);

Description

This QueryAccelType function returns the hardware acceleration framework. See the AccelType definition for details.

Change History

This function was introduced in SDK API 1.0.
QueryDevice

Syntax

```c
pxcI32 QueryDevice(void);
```

Description

This `QueryDevice` function returns the acceleration device index that the SDK acceleration instance works on.

Change History

This function was introduced in SDK API 1.0.

QueryHandle

Syntax

```c
pxcStatus QueryHandle(HandleType type, pxcHDL *handle);
```

Parameters

- `type` The OS handle type; see the `HandleType` enumerator for details.
- `handle` The OS-specific handle, to be returned.

Description

This `QueryHandle` function returns the OS-specific handles that the SDK works on.

Return Status

- `PXC_STATUS_NO_ERROR` The function returned successfully.
- `PXC_STATUS_ITEM_UNAVAILABLE` The specified handle type is not recognized.

Change History

This function was introduced in SDK API 1.0.

SetDevice

Syntax
pxcStatus SetDevice(pxcI32 device);

Parameters

device The acceleration device index

Description

This `SetDevice` function sets the acceleration device.

Return Status

- **pxcStatus_NO_ERROR**: The function returned successfully.
- **pxcStatus_PARAM_UNSUPPORTED**: The specific device is not supported or unable to set the device at this time.

Change History

This function was introduced in SDK API 1.0.

SetHandle

Syntax

pxcStatus SetHandle(HandleType type, pxcHDL handle);

Parameters

type The OS-specific handle type; see the `HandleType` enumerator for details.

handle The OS-specific handle

Description

This `SetHandle` function sets the OS-specific handle that the SDK may work on. The SDK may create internal handles if the application does not provide them explicitly.

Return Status

- **pxcStatus_NO_ERROR**: The function returned successfully.
- **pxcStatus_ITEM_UNAVAILABLE**: The specified handle type is not recognized.

Change History

This function was introduced in SDK API 1.0.
PXCAudio

The **PXCAudio** interface manages the audio buffer access. The audio buffer can be in a system memory buffer or in some OS-specific surfaces. The interface provides a consistent way of accessing the audio buffer.

The application can use the **CreateAudio** function to create an instance of the **PXCAudio** interface.

The **PXCAudio** interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInfo</td>
<td>Return the <strong>AudioInfo</strong> structure.</td>
</tr>
<tr>
<td>QueryOption</td>
<td>Return the audio option flags.</td>
</tr>
<tr>
<td>QueryTimeStamp</td>
<td>Return the time stamp.</td>
</tr>
<tr>
<td>SetOption</td>
<td>Set the audio option flags.</td>
</tr>
<tr>
<td>SetTimeStamp</td>
<td>Set the time stamp.</td>
</tr>
<tr>
<td>CopyData</td>
<td>Copy the audio buffer from another audio object.</td>
</tr>
<tr>
<td>AcquireAccess</td>
<td>Lock the audio buffer for exclusive access, with format conversion.</td>
</tr>
<tr>
<td>TryAccess</td>
<td>Lock the audio buffer for exclusive access, without format conversion.</td>
</tr>
<tr>
<td>ReleaseAccess</td>
<td>Release the lock to the audio buffer.</td>
</tr>
</tbody>
</table>

**AcquireAccess**

**Syntax**

```c
pxcStatus AcquireAccess(Access access, PXCAccelerator *accel, AudioFormat format, AudioData *data);
pxcStatus AcquireAccess(Access access, PXCAccelerator *accel, AudioData *data);
pxcStatus AcquireAccess(Access access, AudioFormat format, AudioData *data);
```
pxcStatus AcquireAccess(Access access, AudioData *data);

Parameters

access
The access type; see the Access enumerator for details.

accel
The optional accelerator instance for the desired output.

format
The optional audio sample format. See the AudioFormat enumerator for details.

data
The output AudioData structure, to be filled.

Description

This function locks the audio buffer for exclusive access. If the specified accelerator or audio format is not available in the audio storage, the function creates an internal copy that matches the accelerator and audio format. The function returns the internal data buffer pointers or OS-specific surface handle in the output AudioData structure.

If the accel instance is omitted, the function uses the system memory accelerator. If the audio format parameter is omitted, the function uses the audio format when the audio buffer was created.

Return Status

PX_C_STATUS_NO_ERROR
The function returned successfully.

PX_C_STATUS_DEVICE_BUSY
Failed to lock the audio buffer for exclusive access.

PX_C_STATUS_PARAM_INVALID
The specified accelerator and format combination are not supported.

Change History

This function was introduced in SDK API 1.0.

CopyData

Syntax

pxcStatus CopyData(PXCAudio *audio);

Parameters

audio
The source audio instance.

Description
This function copies the source audio to this audio buffer.

**Return Status**

| PXC_STATUS_NO_ERROR | The function returned successfully. |

**Change History**

This function was introduced in SDK API 1.0.

---

**QueryInfo**

**Syntax**

```c
pxcStatus QueryInfo(AudioInfo *info);
```

**Parameters**

- `info` The `AudioInfo` structure, to be filled.

**Description**

This function fills the `AudioInfo` structure.

**Return Status**

| PXC_STATUS_NO_ERROR | The function returned successfully. |

**Change History**

This function was introduced in SDK API 1.0.

---

**QueryOption**

**Syntax**

```c
AudioOption QueryOption(void);
```

**Description**

This function returns the audio option flags.

**Change History**

This function was introduced in SDK API 1.0.
QueryTimeStamp

Syntax

pxcU64 QueryTimeStamp(void);

Description

This function returns the image time stamp, in 100 ns.

Change History

This function was introduced in SDK API 1.0.

ReleaseAccess

Syntax

pxcStatus ReleaseAccess(AudioData *data);

Parameters

data

The AudioData structure that is used when the application called the AcquireAccess or the TryAccess functions.

Description

This function releases the lock to the audio storage.

Return Status

PX/status_NO_ERROR The function returned successfully.

Change History

This function was introduced in SDK API 1.0.

SetOption

Syntax

pxcStatus SetOption(AudioOption option);
Parameters

option 

The option flag. See AudioOption enumerator for details.

Description

This function sets the audio option flags.

Return Status

PXC_STATUS_NO_ERROR 

The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

---

SetTimeStamp

Syntax

pxcStatus SetTimeStamp(pxcU64 ts);

Parameters

ts 

The audio time stamp, in 100 ns.

Description

This function sets the audio time stamp.

Return Status

PXC_STATUS_NO_ERROR 

The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

---

TryAccess

Syntax

pxcStatus TryAccess(Access access, PXCAccelerator *accel, AudioFormat format, AudioData *data);

pxcStatus TryAccess(Access access, PXCAccelerator *accel, AudioData *data);


pxcStatus TryAccess(Access access, AudioFormat format, AudioData *data);

pxcStatus TryAccess(Access access, AudioData *data);

Parameters

Access  The access type; see the Access enumerator for details.

Accel  The optional accelerator instance for the desired output.

Format  The optional audio sample format. See the AudioFormat enumerator for details.

Data  The output AudioData structure, to be filled.

Description

This function locks the audio buffer for exclusive access if the audio storage contains a valid copy of audio in the specified accelerator and audio format. The function returns the internal data buffer pointers or OS-specific surface handle in the output AudioData structure.

If the accel instance is omitted, the function uses the system memory accelerator. If the audio format parameter is omitted, the function uses the native audio format.

Return Status

PX_STATUS_NO_ERROR  The function returned successfully.

PX_STATUS_DEVICE_BUSY  Failed to lock the audio buffer for exclusive access.

PX_STATUS_PARAM_INVALID  The specified accelerator and format combination are not supported.

Change History

This function was introduced in SDK API 1.0.

PXCBBase

The PXCBBase interface is the base class of all SDK interface definitions. The interface overrides the class delete operator to work with the SDK dispatching mechanism; and provides a DynamicCast mechanism to replace the dynamic_cast operation, which relies on compiler-specific, run-time type checking. The application that implements any PXCBBase derived interface must derive from the PXCBBaseImpl template class.
The interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamicCast</td>
<td>Query additional interface support</td>
</tr>
<tr>
<td>Release</td>
<td>Release this class instance.</td>
</tr>
</tbody>
</table>

**DynamicCast**

**Syntax**

```c
void *DynamicCast(pxcCUID cuid);
```

**Parameters**

- **cuid**  
The interface identifier. For any SDK interface `X`, the interface identifier is `X::CUID`.

**Description**

This `DynamicCast` function returns the instance of the requested interface if the implementation supports such interface, or `NULL` if the implementation does not support the requested interface.

Note that deleting an instance pointer returned from the `DynamicCast` function completely deletes the instance object. The application must take caution not to double-delete an object.

**Change History**

This function was introduced in SDK API 1.0.

**Remarks**

For simplicity, the application can use the `DynamicCast<T>` template. For example, the following statement casts to a `Y` interface from an `X` interface:

```c
Y *y=x.DynamicCast<Y>();
```

**Release**

**Syntax**

```c
void Release(void);
```
### Description

This **Release** function releases the instance of the [PXCBase](#) interface. As the [PXCBase](#) interface overrode the **delete** operator to call this function, it is equivalent that the application invokes the class destructor or directly calls this function to release the instance.

### Change History

This function was introduced in SDK API 1.0.

### PXCBaseImpl

The PXCBaseImpl template class contains the default implementation of the [PXCBase](#) interface. The application that implements any [PXCBase](#) derived interface, such as a callback handler, should derive the implementation from this template class. See the **SDK Interface** section for details. The PXCBaseImpl template assumes single inheritance. The variations, PXCBaseImpl2 and PXCBaseImpl3, implement two or three inheritances, respectively.

### PXCCallbackBase

The PXCCallbackBase interface is the base class of all SDK callback interface definitions.

The interface exposes the following member functions for backward compatibility reasons. SDK modules do not directly call these functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamicCast</td>
<td>Query additional interface support</td>
</tr>
<tr>
<td>Release</td>
<td>Release this class instance.</td>
</tr>
</tbody>
</table>

### DynamicCast

**Syntax**

```c
void *DynamicCast(pxcCUID cuid);
```

**Parameters**
cuid

The interface identifier. For any SDK interface \( X \), the interface identifier is \( X::\text{CUID} \).

**Description**

This `DynamicCast` function returns the instance of the requested interface if the implementation supports such interface, or `NULL` if the implementation does not support the requested interface.

Note that this function is defined for backward compatibility reasons. SDK modules do not directly call this function.

**Change History**

This function was introduced in SDK API 1.0.

**Remarks**

The application can use the `DynamicCast<T>` template to enable type checking. For example, the following statement casts to a `Y` interface from an `X` interface:

\[
Y \ast y=x\rightarrow\text{DynamicCast}<Y>();
\]

**Release**

**Syntax**

```c
void Release(void);
```

**Description**

This `Release` function destroys the callback instance.

Note that this function is defined for backward compatibility reasons. SDK modules do not directly call this function.

**Change History**

This function was introduced in SDK API 1.0.

**PXCCapture**

The `PXCCapture` interface provides member functions to query video capture devices and create an instance of the capture devices. The application can create the `PXCCapture` interface using the `CreateImpl` function with the following module descriptor values:
The PXCCapture interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryDevice</td>
<td>Query information of a capture device by index.</td>
</tr>
<tr>
<td>CreateDevice</td>
<td>Create an instance of the capture device.</td>
</tr>
</tbody>
</table>

### CreateDevice

**Syntax**

```c
pxcStatus CreateDevice(pxCU32 didx, Device **device);
```

**Parameters**

- **didx**
  
  The zero-based capture device index.

- **device**
  
  The capture device instance to be returned.

**Description**

This `CreateDevice` function creates an instance of the `Device` interface instance for the given capture device.

**Return Status**

- `PXC_STATUS_NO_ERROR`
  
  The function completed successfully.

- `PXC_STATUS_ITEM_UNAVAILABLE`
  
  No capture device was found with the given index.

**Change History**

This function was introduced in SDK API 1.0.

### QueryDevice

**Syntax**

```c
pxcStatus QueryDevice(pxCU32 didx, DeviceInfo *dinfo);
```
Parameters

- *didx*  
  Zero-based capture device index.

- *dinfo*  
  The device information to be returned.

Description

This **QueryDevice** function returns the capture device information by index. The number of available capture devices and their order may change over time, thus the application should not assume any particular value of the capture device index, or order. The SDK enumerates available capture devices in the system when the application creates the **PXCCapture** instance.

Return Status

- **PXC_STATUS_NO_ERROR**  
  The function completed successfully.

- **PXC_STATUS_ITEM_UNAVAILABLE**  
  No capture device found with the given index.

Change History

This function was introduced in SDK API 1.0.

---

**PXCCapture::AudioStream**

The **AudioStream** interface manages the audio stream profile information and reading samples from the audio capture device. The application can create this interface by using the **QueryAudioStream** function.

The **AudioStream** interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QueryStream</strong></td>
<td>Return the stream information.</td>
</tr>
<tr>
<td><strong>QueryProfile</strong></td>
<td>Return the stream configuration information.</td>
</tr>
<tr>
<td><strong>SetProfile</strong></td>
<td>Set the stream configuration parameters.</td>
</tr>
<tr>
<td><strong>ReadStreamAsync</strong></td>
<td>Read an audio frame from the capture device.</td>
</tr>
</tbody>
</table>
### QueryProfile

**Syntax**

```c
pxcStatus QueryProfile(pxcU32 pidx, ProfileInfo *pinfo);
pxcStatus QueryProfile(ProfileInfo *pinfo);
```

**Parameters**

- `pidx`  
  The zero-based profile index. Use WORKING_PROFILE, or the `pidx` omitted version of the function, to retrieve the current working parameter set.

- `pinfo`  
  The configuration information to be returned.

**Description**

This function returns the configuration parameters for the specified profile index.

**Return Status**

- `PXC_STATUS_NO_ERROR`  
  The function returned successfully.

- `PXC_STATUS_ITEM_UNAVAILABLE`  
  The profile index exceeds its range.

**Change History**

This function was introduced in SDK API 1.0.

### QueryStream

**Syntax**

```c
pxcStatus QueryStream(StreamInfo *sinfo);
```

**Parameters**

- `sinfo`  
  The `StreamInfo` structure to be returned.

**Description**

This function returns the stream information.

**Return Status**

- `PXC_STATUS_NO_ERROR`  
  The function returned successfully.
Change History

This function was introduced in SDK API 1.0.

ReadStreamAsync

Syntax

```c
pxcStatus ReadStreamAsync(PXCAudio **audio, PXCScheduler::SyncPoint **sp);
```

Parameters

- `audio` The PXCAudio structure to retrieve the audio frame.
- `sp` The SP to be returned.

Description

This function reads an audio frame from the audio capture device. The function allocates the audio buffer. The application needs to release the buffer.

If there is a timestamp gap in delivering the video sequences, the function returns `PXC_STATUS_TIME_GAP`.

This function is an asynchronous function. The application must synchronize the SP before accessing the captured audio frame.

Return Status

- `PXC_STATUS_NO_ERROR` The function returned successfully.
- `PXC_STATUS_TIME_GAP` There is a gap in the video sequence.

Change History

This function was introduced in SDK API 1.0.

SetProfile

Syntax

```c
pxcStatus SetProfile(ProfileInfo *pinfo);
```

Parameters
**Pinfo**

The specified profile parameters.

**Description**

This function configures the video capture device with the specified profile parameters. The profile parameters do not have to be exactly the same as the device returns if the device supports configuration parameter conversion.

**Return Status**

- **PXC_STATUS_NO_ERROR**
  The function returned successfully.
- **PXC_STATUS_PARAM_UNSUPPORTED**
  Unsupported parameters in the configuration.
- **PXC_STATUS_PARAM_INPLACE**
  The device is already configured with the same parameters.

**Change History**

This function was introduced in SDK API 1.0.

**PXCCapture::Device**

The **Device** interface provides member functions to enumerate capture video device stream information and create an instance of a video stream. The application can create this interface using the **CreateDevice** function.

The **Device** interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QueryDevice</strong></td>
<td>Return the capture device information.</td>
</tr>
<tr>
<td><strong>QueryStream</strong></td>
<td>Return the capture device stream information.</td>
</tr>
<tr>
<td><strong>CreateStream</strong></td>
<td>Create a stream interface instance.</td>
</tr>
<tr>
<td><strong>QueryPropertyInfo</strong></td>
<td>Return the information about a device property, such as the range, the default value, and whether it is a manual or automatic control.</td>
</tr>
<tr>
<td><strong>QueryProperty</strong></td>
<td>Return the property value as a single value.</td>
</tr>
<tr>
<td><strong>SetPropertyAuto</strong></td>
<td>Configure the device by setting the specified property to automatic control.</td>
</tr>
</tbody>
</table>
**SetProperty**

Configure the device by setting the specified property to the specified value.

The **Device** interface exposes the following constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCAP_LIMIT</td>
<td>The maximum number of device properties for a device.</td>
</tr>
</tbody>
</table>

**CreateStream**

**Syntax**

```c
pxcStatus CreateStream(pxcU32 sidx, pxCUID cuid, void **stream);
```

**Parameters**

- `sidx` The zero-based stream index.
- `cuid` The interface identifier to be created.
- `stream` The video stream interface instance to be returned.

**Description**

This function creates an instance of the specified stream. Do not create multiple instances of the same stream.

For image/depth streams, the returned stream is an instance of the `PXCCapture::VideoStream` interface. Use `cuid=PXCCapture::VideoStream::CUID`.

For audio streams, the returned stream is an instance of the `PXCCapture::AudioStream` interface. Use `cuid=PXCCapture::AudioStream::CUID`.

**Return Status**

- `PXC_STATUS_NO_ERROR` The function completed successfully.
- `PXC_STATUS_ITEM_UNAVAILABLE` No stream found with the given index.

**Change History**

This function was introduced in SDK API 1.0.

**Remarks**

The application can use the `CreateStream<T>` template to enable type checking. For
example, the following statement creates an instance (in variable $y$) of stream type $Y$:

```c
pxcStatus sts=device->CreateStream<Y>(sidx, &y);
```

### QueryDevice

**Syntax**

```c
pxcStatus QueryDevice(DeviceInfo *dinfo);
```

**Parameters**

- `dinfo` The capture device information to be returned.

**Description**

This function returns the capture device information.

**Return Status**

- `PXC_STATUS_NO_ERROR` The function completed successfully.

**Change History**

This function was introduced in SDK API 1.0.

### QueryPropertyInfo

**Syntax**

```c
pxcStatus QueryPropertyInfo(Property property, PXCRangeF32 *range, pxcF32 *step, pxcF32 *default, pxcBool *isAuto);
```

**Parameters**

- `property` The device property; see the `Property` enumerator for definitions.
- `range` The range of the control parameter.
- `step` The step of the control parameter.
- `default` The default value of the control parameter.
- `isAuto` Boolean whether the control is auto controlled.
Description

This function returns the information of the device property such as range, default value, etc.

Return Status

PXC_STATUS_NO_ERROR          The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE  Failed to locate the property.

Change History

This function was introduced in SDK API 1.0.

QueryProperty

Syntax

```c
pxcStatus QueryProperty(Property property, pxcF32 *value);
```

Parameters

- **property**: The device property; see the `Property` enumerator for definitions.
- **value**: The property value, to be returned.

Description

This function returns the current property value.

Return Status

PXC_STATUS_NO_ERROR          The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE  Failed to locate the property.

Change History

This function was introduced in SDK API 1.0.

Remarks

The following variations are provided as convenience:

```c
pxcStatus QueryPropertyAs3DPoint(Property property, PXCPoint3DF32 *point);
pxcStatus QueryPropertyAsPoint(Property property, PXCF32 *point);
```
### QueryStream

**Syntax**

```c
pxcStatus QueryStream(pxcU32 sidx, StreamInfo *sinfo);
```

**Parameters**

- `sidx` The zero-based stream index.
- `sinfo` The stream information to be returned.

**Description**

This function returns the stream information given the stream index.

**Return Status**

- `PXC_STATUS_NO_ERROR` The function completed successfully.
- `PXC_STATUS_ITEM_UNAVAILABLE` No stream found with the given index.

**Change History**

This function was introduced in SDK API 1.0.

### SetPropertyAuto

**Syntax**

```c
pxcStatus SetPropertyAuto(Property property, pxcBool ifauto);
```

**Parameters**

- `property` The device property; see the `Property` enumerator for definitions.
- `ifauto` Boolean value to indicate if automatic control is enabled.
Description

This function enables/disables automatic control of the device property if the property supports automatic control.

The application must first acquire exclusive access to the device before configuring the device.

Return Status

- PXC_STATUS_NO_ERROR: The function completed successfully.
- PXC_STATUS_ITEM_UNAVAILABLE: Failed to locate the property.
- PXC_STATUS_ACCESS_DENIED: Failed to set automatic control.

Change History

This function was introduced in SDK API 1.0.

SetProperty

Syntax

pxcStatus SetProperty(Property property, pxcF32 value);

Parameters

- property: The device property; see the Property enumerator for definitions.
- value: The desired value.

Description

This function sets the property value.

The application must first acquire exclusive access to the device before configuring the device.

Return Status

- PXC_STATUS_NO_ERROR: The function completed successfully.
- PXC_STATUS_ITEM_UNAVAILABLE: Failed to locate the device property.
- PXC_STATUS_ACCESS_DENIED: Failed to set the value.

Change History
This function was introduced in SDK API 1.0.

**PXCCapture::VideoStream**

The VideoStream interface manages the video stream profile information and reading samples from the video capture device. The application can create this interface using the QueryVideoStream function.

The VideoStream interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryStream</td>
<td>Return the stream information.</td>
</tr>
<tr>
<td>QueryProfile</td>
<td>Return the stream configuration information.</td>
</tr>
<tr>
<td>SetProfile</td>
<td>Set the stream configuration parameters.</td>
</tr>
<tr>
<td>ReadStreamAsync</td>
<td>Read an image or video frame from the capture device.</td>
</tr>
</tbody>
</table>

The VideoStream interface exposes the following constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM_LIMIT</td>
<td>The maximum number of streams that can be specified in the DataDesc structure.</td>
</tr>
</tbody>
</table>

**QueryProfile**

**Syntax**

```c
pxcStatus QueryProfile(pxcU32 pidx, ProfileInfo *pinfo);
pxcStatus QueryProfile(ProfileInfo *pinfo);
```

**Parameters**

- `pxcU32 pidx` The zero-based profile index. Use WORKING_PROFILE, or the timeout-omitted function, to retrieve the current working parameter set.
- `ProfileInfo *pinfo` The profile information to be returned.
Description

This function returns the profile information with the given profile index.

Return Status

- **PXC_STATUS_NO_ERROR** The function returned successfully.
- **PXC_STATUS_ITEM_UNAVAILABLE** The profile index exceeds its range.

Change History

This function was introduced in SDK API 1.0.

---

**QueryStream**

**Syntax**

```c
pxcStatus QueryStream(StreamInfo *sinfo);
```

**Parameters**

- **sinfo** The `StreamInfo` structure to be returned.

**Description**

This function returns the stream information.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function returned successfully.

**Change History**

This function was introduced in SDK API 1.0.

---

**ReadStreamAsync**

**Syntax**

```c
pxcStatus ReadStreamAsync(PXCImage **image, PXCScheduler::SyncPoint **sp);
```

**Parameters**
audio

The **PXCImage** structure to retrieve the image or video frame.

sp

The SP to be returned.

**Description**

This function reads an image or video frame from the video capture device. The function allocates the image buffer. The application needs to release the buffer.

If there is a timestamp gap in delivering the video sequences, the function returns **PXC_STATUS_TIME_GAP**.

This function is an asynchronous function. The application must synchronize the SP before accessing the captured image or video frame.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function returned successfully.
- **PXC_STATUS_TIME_GAP** There is a gap in the video sequence.

**Change History**

This function was introduced in SDK API 1.0.

---

**SetProfile**

**Syntax**

```c
pxcStatus SetProfile(ProfileInfo *pinfo);
```

**Parameters**

- **pinfo** The specified profile parameters.

**Description**

This function configures the video capture device with the specified profile parameters. The profile parameters do not have to be exactly the same as the device returns if the device supports configuration conversion.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function returned successfully.
- **PXC_STATUS_PARAM_UNSUPPORTED** Unsupported parameters in the configuration.
- **PXC_STATUS_PARAM_INPLACE** The device is already configured with the same parameters.
Change History

This function was introduced in SDK API 1.0.

PXCImage

The PXCImage interface manages image buffer access. The image buffer can be in a system memory buffer or in some OS-specific surface. The interface provides a consistent way of accessing the image buffer.

The application use the CreateImage function to create an instance of the PXCImage interface.

The PXCImage interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryInfo</td>
<td>Return the image properties.</td>
</tr>
<tr>
<td>QueryOption</td>
<td>Return any optional flags.</td>
</tr>
<tr>
<td>SetOption</td>
<td>Set the optional flags.</td>
</tr>
<tr>
<td>QueryTimeStamp</td>
<td>Return the time stamp.</td>
</tr>
<tr>
<td>SetTimeStamp</td>
<td>Set the time stamp.</td>
</tr>
<tr>
<td>QueryROI</td>
<td>Return the region of interest.</td>
</tr>
<tr>
<td>SetROI</td>
<td>Set the region of interest.</td>
</tr>
<tr>
<td>CopyData</td>
<td>Copy an external image to the image.</td>
</tr>
<tr>
<td>AcquireAccess</td>
<td>Lock the image storage for read/write access with format conversion.</td>
</tr>
<tr>
<td>TryAccess</td>
<td>Lock the image storage for read/write access without format conversion.</td>
</tr>
<tr>
<td>ReleaseAccess</td>
<td>Unlock the image storage.</td>
</tr>
</tbody>
</table>

AcquireAccess

Syntax
pxcStatus AcquireAccess(Access access, PXCAccelerator *accel, ColorFormat format, ImageData *data);

pxcStatus AcquireAccess(Access access, PXCAccelerator *accel, ImageData *data);

pxcStatus AcquireAccess(Access access, ColorFormat format, ImageData *data);

pxcStatus AcquireAccess(Access access, ImageData *data);

Parameters

access
The access type; see the Access enumerator for details.

accel
Optional accelerator instance for the required output.

format
Optional color format of required buffer. See the ColorFormat enumerator for details.

data
The output ImageData structure, to be filled.

Description

This function locks the image buffer for exclusive access. If the specified accelerator or color format is not available for the image storage, the function creates an internal copy that matches the accelerator and color format. The function returns the internal data buffer pointers or OS-specific surface handle in the output ImageData structure.

If the accel parameter is NULL or omitted, the function uses the system accelerator. If the format parameter is NULL or omitted, the function returns the color format that the image was created.

Return Status

PXC_STATUS_NO_ERROR
The function returned successfully.

PXC_STATUS_DEVICE_BUSY
Failed to lock the image buffer for exclusive access.

PXC_STATUS_PARAM_INVALID
The specified accelerator and color combination are not supported.

Change History

This function was introduced in SDK API 1.0.

CopyData

Syntax
```c
pxcStatus CopyData(PXCIImage *image);
```

**Parameters**

- `image`: The source image.

**Description**

This function copies the source image to the image storage. The source may be of a different surface type than this image surface type. The SDK does limited surface type conversion. The SDK does not perform any image format conversion.

**Return Status**

- `PXC_STATUS_NO_ERROR`: The function returned successfully.
- `PXC_STATUS_PARAM_UNSUPPORTED`: Failed to copy image due to unsupported image/format conversion.

**Change History**

This function was introduced in SDK API 1.0.

---

**QueryInfo**

**Syntax**

```c
pxcStatus QueryInfo(ImageInfo *info);
```

**Parameters**

- `info`: The `ImageInfo` structure to be returned.

**Description**

This function returns the image properties.

**Change History**

This function was introduced in SDK API 1.0.

---

**QueryOption**

**Syntax**

```c
ImageOption QueryOption(void);
```
**Description**

This function returns the image option flags.

**Change History**

This function was introduced in SDK API 1.0.

---

### QueryROI

**Syntax**

```c
PXCRectU32 QueryROI(void);
```

**Description**

This function returns the image region of interest in pixels.

**Change History**

This function was introduced in SDK API 1.0.

---

### QueryTimeStamp

**Syntax**

```c
pxcU64 QueryTimeStamp(void);
```

**Description**

This function returns the image time stamp, in 100 ns.

**Change History**

This function was introduced in SDK API 1.0.

---

### ReleaseAccess

**Syntax**

```c
pxcStatus ReleaseAccess(ImageData *data);
```

**Parameters**
The **ImageData** structure that is used when the application called the **AcquireAccess** function.

**Description**

This function releases the lock to the image storage.

**Return Status**

```
PX_C_STATUS_NO_ERROR  The function returned successfully.
```

**Change History**

This function was introduced in SDK API 1.0.

---

### SetOption

**Syntax**

```
pxcStatus SetOption(ImageOption option);
```

**Parameters**

- `option` The image options. See the **ImageOption** enumerator for details.

**Description**

This function sets the image options.

**Return Status**

```
PX_C_STATUS_NO_ERROR  The function completed successfully.
```

**Change History**

This function was introduced in SDK API 1.0.

---

### SetROI

**Syntax**

```
pxcStatus SetROI(PXCRectU32 roi);
```

**Parameters**
roi

The image region of interest in pixel.

**Description**

This function sets the image region of interest.

**Return Status**

PXC_STATUS_NO_ERROR  The function completed successfully.

**Change History**

This function was introduced in SDK API 1.0.

---

**SetTimeStamp**

**Syntax**

```c
pxcStatus SetTimeStamp(pxcU64 ts);
```

**Parameters**

- `ts` The image time stamp, in 100 ns.

**Description**

This function sets the image time stamp.

**Return Status**

PXC_STATUS_NO_ERROR  The function completed successfully.

**Change History**

This function was introduced in SDK API 1.0.

---

**TryAccess**

**Syntax**

```c
pxcStatus TryAccess(Access access, PXCAccelerator *accel, ColorFormat format, ImageData *data);
```

```c
pxcStatus TryAccess(Access access, PXCAccelerator *accel, ImageData *data);
```

```c
pxcStatus TryAccess(Access access, ColorFormat format, ImageData *data);
```
pxcStatus TryAccess(Access access, ImageData *data);

Parameters

access The access type; see the Access enumerator for details.
accel Optional accelerator instance for the required output.
format Optional color format of required buffer. See the ColorFormat enumerator for details.
data The output ImageData structure, to be filled.

Description

This function locks the image buffer for exclusive access if the image storage contains a valid copy of the image in the specified accelerator and color format. The function returns the internal data buffer pointers or OS-specific surface handle in the output ImageData structure.

If the accel parameter is NULL or omitted, the function uses the system accelerator. If the format parameter is NULL or omitted, the function returns the color format that the image was created.

Return Status

PXC_STATUS_NO_ERROR The function returned successfully.
PXC_STATUS_DEVICE_BUSY Failed to lock the image buffer for exclusive access.
PXC_STATUS_PARAM_INVALID The specified accelerator and color combination are not valid or supported.

Change History

This function was introduced in SDK API 1.0.

PXCMetadata

The PXCMetadata interface manages the metadata storage. The PXCSession, PXImage and PXCAudio implementations expose the PXCMetadata interface.

The PXCMetadata interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryUID</td>
<td>Return an unique identifier for metadata identification</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>QueryMetadata</td>
<td>Enumerate all metadata identifiers</td>
</tr>
<tr>
<td>DetachMetadata</td>
<td>Detach meta data from the metadata storage</td>
</tr>
<tr>
<td>AttachBuffer</td>
<td>Attach a byte buffer to the metadata storage</td>
</tr>
<tr>
<td>QueryBuffer</td>
<td>Retrieve the byte buffer from the metadata storage</td>
</tr>
<tr>
<td>AttachSerializable</td>
<td>Attach a serializable interface implementation</td>
</tr>
<tr>
<td>CreateSerializable</td>
<td>Recreate the interface implementation from the metadata storage</td>
</tr>
</tbody>
</table>

**AttachBuffer**

**Syntax**

```c
pxcStatus AttachBuffer(pxcUID id, pxcBYTE *buffer, pxcU32 size);
```

**Parameters**

- `id` The metadata identifier
- `buffer` The pointer to the byte buffer
- `size` The size of the byte buffer

**Description**

This function attaches a byte buffer to the metadata storage.

**Return Status**

- `PXC_STATUS_NO_ERROR` The function completed successfully.

**Change History**

This function was introduced in SDK API 1.0.

**AttachSerializable**

**Syntax**

```c
pxcStatus AttachSerializable(pxcUID id, PXCBASE *serializable);
```

**Parameters**
id                  The metadata identifier
serializable       The instance of the serializable interface

Description
This function serializes the interface implementation and attaches the serialized data to the metadata storage. The application can later recreate the interface implementation from the serialized data.

Return Status
PXC_STATUS_NO_ERROR          The function completed successfully.

Change History
This function was introduced in SDK API 1.0.

CreateSerializable

Syntax
pxcStatus CreateSerializable(pxCUID id, pxCUID cuid, void **serializable);

Parameters
id                  The metadata identifier
cuid                The interface identifier
serializable       The instance of the serializable interface, to be created.

Description
This function recreates the interface implementation from the serialized data in the metadata storage.

Return Status
PXC_STATUS_NO_ERROR          The function completed successfully.

Change History
This function was introduced in SDK API 1.0.

Remarks
The application may use the `CreateSerialize<T>` template to enable type checking. For example, the following statement recreates the interface `Y` implementation in variable `y`:

```cpp
pxcStatus sts=metadata->CreateSerializable<Y>(id,&y);
```

### DetachMetadata

**Syntax**

```cpp
pxcStatus DetachMetadata(pxcUID id);
```

**Parameters**

- `id` The metadata identifier

**Description**

This function detaches the identified metadata.

**Return Status**

- `PXC_STATUS_NO_ERROR` The function completed successfully.
- `PXC_STATUS_ITEM_UNAVAILABLE` The metadata does not exist.

**Change History**

This function was introduced in SDK API 1.0.

### QueryBuffer

**Syntax**

```cpp
pxcStatus QueryBuffer(pxcUID id, pxcBYTE *buffer, pxcU32 *size);
```

**Parameters**

- `id` The metadata identifier
- `buffer` Optional pointer to the byte buffer
- `size` Optional pointer to retrieve the buffer size

**Description**
This function retrieves the byte buffer content from the metadata storage. The application must allocate the buffer. To retrieve the buffer size, the application can call this function with a NULL buffer pointer.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function completed successfully.
- **PXC_STATUS_ITEM_UNAVAILABLE** The metadata does not exist.

**Change History**

This function was introduced in SDK API 1.0.

---

### QueryMetadata

**Syntax**

```c
pxcStatus QueryMetadata(pxcU32 idx, pxCUID *id);
```

**Parameters**

- **idx** Zero-based index to enumerate all metadata identifiers.
- **id** The metadata identifier, to be returned.

**Description**

This function enumerates all available metadata identifiers by returning the metadata identifier of the given index.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function completed successfully.
- **PXC_STATUS_ITEM_UNAVAILABLE** The metadata does not exist.

**Change History**

This function was introduced in SDK API 1.0.

---

### QueryUID

**Syntax**

---
pxcUID QueryUID(void);

Parameters

None

Description

This function returns an unique identifier for next metadata item, or zero if the operation is failed.

Change History

This function was introduced in SDK API 1.0.

## PXCPowerState

The PXCPowerState interface manages the SDK implementation power state. Any SDK I/O or algorithm module implementation that are power aware exposes this interface. The application can use the DynamicCast function to query this interface from any module instance.

The PXCPowerState interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryState</td>
<td>Query the current power state.</td>
</tr>
<tr>
<td>setState</td>
<td>Set the power state.</td>
</tr>
</tbody>
</table>

### QueryState

Syntax

```c
pxcStatus QueryState(State *state, State *istate);
```

Parameters

- **state**
  
  The current power state mode, to be returned. See the State enumerator for details.

- **istate**
  
  The optional pointer to return the actual power state. See the State enumerator for details.
Description

This function returns the current power state that a SDK module is in. For power state `STATE_ADAPTIVE`, the parameter `state` returns `STATE_ADAPTIVE` and the parameter `istate` returns the actual power state, one of `STATE_C0` to `STATE_C3`. For other power states, the parameters `state` and `istate` return the same value.

Return Status

- `PXC_STATUS_NO_ERROR` The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

SetState

Syntax

```c
pxcStatus SetState(State state);
```

Parameters

- `state` The desired power state. See the `State` enumerator for details.

Description

This function sets the module power state.

Return Status

- `PXC_STATUS_NO_ERROR` The function completed successfully.
- `PXC_STATUS_PARAM_UNSUPPORTED` The module does not support the specified power state.

Change History

This function was introduced in SDK API 1.0.

PXCPrediction

The `PXCPrediction` interface provides functions to project or map among color, depth and real world coordinates. The `PXCPrediction` implementation is usually provided by an I/O module as
a serializable implementation attached to the session metadata. For example, the following statements create a PXCProjection instance from PXCCapture::Device:

\[
\text{device} \rightarrow \text{QueryPropertyAsUID}(	ext{PXCCapture::Device::PROPERTY_PROJECTION_SERIALIZABLE}, \& \text{id})
\]

\[
\text{session} \rightarrow \text{DynamicCast<PXCMetadata>() \rightarrow CreateSerializable<PXCProjection>(id, \&projection)}
\]

The PXCProjection interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapColorToDepthCoordinates</td>
<td>Map color coordinates to depth coordinates.</td>
</tr>
<tr>
<td>MapDepthToColorCoordinates</td>
<td>Map depth coordinates to color coordinates.</td>
</tr>
<tr>
<td>ProjectImageToRealWorld</td>
<td>Project image coordinates to real world coordinates.</td>
</tr>
<tr>
<td>ProjectRealWorldToImage</td>
<td>Project real world coordinates to image coordinates.</td>
</tr>
</tbody>
</table>

### MapColorToDepthCoordinates

**Syntax**

\[
\text{pxcStatus} \ \text{MapColorToDepthCoordinates}(\text{pxcU32 npoints, PXPointF32 *posc, PXPointF32 *posd});
\]

**Parameters**

- npoints: Number of points to map.
- posc: An array of color coordinates
- posd: An array of depth coordinates, to be mapped.

**Description**

This function maps an array of color coordinates to depth coordinates.

**Return Status**

- \text{PXC_STATUS_NO_ERROR} The function completed successfully.

**Change History**

This function was introduced in SDK API 1.0.
MapDepthToColorCoordinates

Syntax

```c
pxcStatus MapDepthToColorCoordinates(pxcU32 npoints, PXCPaintPoint3D *posd, PXCPaintPointF *posc);
```

Parameters

- **npoints**: Number of points to map.
- **posd**: An array of depth coordinates in the PXCPaintPoint3D structure. The application must fill the z field with the depth value.
- **posc**: An array of color coordinates, to be mapped.

Description

This function maps an array of depth coordinates to color coordinates. If the function failed to map certain coordinates, for example, some depth value is floored or saturated, the function returns (-1,-1) in the color coordinates to indicate the failure.

Return Status

- **PXC_STATUS_NO_ERROR**: The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

ProjectImageToRealWorld

Syntax

```c
pxcStatus ProjectImageToRealWorld(pxcU32 npoints, PXCPaintPoint3D *pos2d, PXCPaintPoint3DF *pos3d);
```

Parameters

- **npoints**: Number of points to map.
- **pos2d**: An array of depth coordinates in the PXCPaintPoint3D structure. The application must fill the z field with the depth value.
- **pos3d**: An array of 3D coordinates, to be mapped.
pos3d

An array of real world coordinates, to be mapped.

Description

This function projects depth image coordinates to real world coordinates.

Return Status

PXC_STATUS_NO_ERROR The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

ProjectRealWorldToImage

Syntax

pxcStatus ProjectRealWorldToImage(pxcU32 npoints, PXPoint3DF32 *pos3d, PXPointF32 *pos2d);

Parameters

npoints Number of points to map.
pos3d An array of real world coordinates.
pos2d An array of depth image coordinates, to be projected.

Description

This function projects an array of real world coordinates to depth image coordinates.

Return Status

PXC_STATUS_NO_ERROR The function completed successfully.

Change History

This function was introduced in SDK API 1.0.
The **PXCScheduler** interface manages asynchronous pipeline execution. The application uses the `CreateScheduler` function to create an instance of the **PXCScheduler** interface. The application can create multiple schedulers with different priorities. Internally, the SDK coordinates all execution requests according to the specified priorities. If not specified, the SDK session uses its default scheduler, which is normal execution priority.

The **PXCScheduler** interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryPriority</td>
<td>Query the scheduler priority.</td>
</tr>
<tr>
<td>SetPriority</td>
<td>Set the scheduler priority.</td>
</tr>
</tbody>
</table>

### QueryPriority

**Syntax**

```c
pxcStatus QueryPriority(Priority *priority);
```

**Parameters**

| Priority          | The scheduler priority, to be returned; see the `Priority` enumerator for details. |

**Description**

This function returns the SDK scheduler priority.

**Return Status**

| PXC_STATUS_NO_ERROR | The function completed successfully. |

**Change History**

This function was introduced in SDK API 1.0.

### SetPriority

**Syntax**

```c
pxcStatus SetPriority(Priority priority);
```
Parameters

Priority

The SDK scheduler priority; see the Priority enumerator for details.

Description

This function sets the SDK scheduler priority.

Return Status

PXC_STATUS_NO_ERROR

The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

PXCScheduler::SyncPoint

The SyncPoint interface manages SP synchronization. The application can query the status of an execution as well as wait for the asynchronous execution to complete. The SP is created by each SDK asynchronous function. The application must delete the SP after use.

The SyncPoint interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryPtr</td>
<td>Query the execution status of a single output.</td>
</tr>
<tr>
<td>Synchronize</td>
<td>Synchronize a single asynchronous operation or an asynchronous pipeline.</td>
</tr>
<tr>
<td>SynchronizeEx</td>
<td>Synchronize multiple asynchronous operations or asynchronous pipelines.</td>
</tr>
</tbody>
</table>

The SyncPoint interface exposes the following constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEOUT_INFINITE</td>
<td>Wait infinitely for synchronization.</td>
</tr>
<tr>
<td>SYNCEX_LIMIT</td>
<td>The maximum number of SPs that the SynchronizeEx function can wait on.</td>
</tr>
</tbody>
</table>
QueryPtr

Syntax

```c
pxcStatus QueryPtr(void *ptr);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>The asynchronous output to be queried.</td>
</tr>
</tbody>
</table>

Description

This `QueryPtr` function returns the execution status of a particular asynchronous output.

Return Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXC_STATUS_EXECUTION_ABORTED</td>
<td>The asynchronous operation is aborted due to upstream asynchronous operations.</td>
</tr>
<tr>
<td>PXC_STATUS_EXECUTION_INPROGRESS</td>
<td>The asynchronous operation is ongoing.</td>
</tr>
<tr>
<td>Other values</td>
<td>The return status of the asynchronous operation that generates the specified output.</td>
</tr>
</tbody>
</table>

Change History

This function was introduced in SDK API 1.0.

Synchronize

Syntax

```c
pxcStatus Synchronize(pxcU32 timeout);
pxcStatus Synchronize(void);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The time out value in milliseconds. Use TIMEOUT_INFINITE, or the timeout omitted function to wait infinitely.</td>
</tr>
</tbody>
</table>

Description

This `Synchronize` function waits until an asynchronous execution is completed, or returns due to time out.
Return Status

**PXC_STATUS_EXEC_TIMEOUT**

The asynchronous operation is still running. The function returns due to time out.

Other values

The return status of the completed asynchronous operation.

Change History

This function was introduced in SDK API 1.0.

### SynchronizeEx

**Syntax**

```c
pxcStatus SynchronizeEx(pxcU32 n1, SyncPoint **sps, pxcU32 n2, pxcHDL *objs, pxcU32 *idx, pxcU32 timeout);
pxcStatus SynchronizeEx(pxcU32 n1, SyncPoint **sps, pxcU32 *idx, pxcU32 timeout);
pxcStatus SynchronizeEx(pxcU32 n1, SyncPoint **sps, pxcU32 *idx);
pxcStatus SynchronizeEx(pxcU32 n1, SyncPoint **sps);
```

**Parameters**

- **n1**
  
  The number of synchronization points.

- **sps**
  
  The synchronization point array. The function skips any invalid SPs (zero value).

- **n2**
  
  The number of OS-specific synchronization objects.

- **objs**
  
  An optional array of OS-specific synchronization objects. In Windows*, the objects can be mutexes, semaphores, events, or thread handles.

- **idx**
  
  An optional pointer to return the completed sync point index; if the synchronization points signal, the index is from 0 to n1−1. If the OS-specific synchronization objects signal, the index is from n1 to n1+n2−1.

- **timeout**
  
  The time out value in milliseconds. Use **TIMEOUT_INFINITE**, or the timeout omitted functions, to wait infinitely.

**Description**
This `SynchronizeEx` function waits until one or all of the asynchronous executions completes, or returns due to time out. If `idx! = 0`, the function returns when any of the asynchronous operations is completed.

If more than one synchronization objects signal, the function returns only the first signaled synchronization object. The application may need to loop through all synchronization objects to see which one signaled.

There is a limit how many SPs the function can wait on. The limit is specified as `SYNCEX_LIMIT`, and is OS-specific.

**Return Status**

- **PXC_STATUS_PARAM_UNSUPPORTED** The number of synchronization objects exceeds OS limitation.
- **PXC_STATUS_TIMEOUT** The asynchronous operation is still running. The function returns due to time out.
- **PXC_STATUS_NO_ERROR** All asynchronous operations are completed. Check each synchronization point for completed return status.
- **Other values** If one of the synchronization points (`SyncPoint`) signals, the function returns the status of the synchronization.

**Change History**

This function was introduced in SDK API 1.0.

---

**PXCSession**

The **PXCSession** interface maintains the SDK context. The application can query and create instances of I/O module and algorithm module implementations, as well as critical `PXCScheduler` and `PXCAccelerator` objects. The **PXCSession** implementation exposes `PXCMetadata` interface for centralized metadata storage.

The application uses the **PXCSession_Create** function to create an instance of the PXCSession interface.

The **PXCSession** interface exposes the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>QueryVersion</code></td>
<td>Initialize an SDK session.</td>
</tr>
<tr>
<td><code>QueryImpl</code></td>
<td>Query an implementation.</td>
</tr>
</tbody>
</table>
CreateScheduler

Create a scheduler instance.

CreateAccelerator

Create an accelerator instance.

CreateImpl

Create a module instance.

LoadImplFromFile

Add a module shared library to the session.

UnloadImplFromFile

Remove a module shared library from the session.

---

CreateAccelerator

Syntax

```cpp
pxcStatus CreateAccelerator(PXCAccelerator::AccelType type, PXCAccelerator **instance);
pxcStatus CreateAccelerator(PXCAccelerator **instance);
```

Parameters

- `type`: An optional parameter to specify the acceleration framework; see the `AccelType` enumerator for details.
- `instance`: Return the instance of the `PXCAccelerator` interface.

Description

This `CreateAccelerator` function creates an instance of the `PXCAccelerator` interface. If `device` and `type` are not specified, the function creates a CPU accelerator.

Return Status

- `PXC_STATUS_NO_ERROR`: The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

CreateImpl

Syntax

```cpp
pxcStatus CreateImpl(PXCScheduler *scheduler, PXCAccelerator *accelerator, ImplDesc *desc, pxcUID iuid, pxcUID cuid, void **instance);
```
pxcStatus CreateImpl(PXCScheduler *scheduler, PXCAccelerator *accelerator, ImplDesc *desc, pxcUID cuid, void **instance);
pxcStatus CreateImpl(pxCUID iuid, pxCUID cuid, void **instance);
pxcStatus CreateImpl(pxCUID cuid, void **instance);
pxcStatus CreateImpl(ImplDesc *desc, pxCUID cuid, void **instance);

Parameters

scheduler Optional SDK scheduler instance; if zero, the SDK uses the default scheduler.
accelerator Optional SDK accelerator instance; if zero, the SDK uses the CPU accelerator.
desc Optional module descriptor template; see the ImplDesc structure for details.
iuid The module implementation identifier.
cuid The module interface identifier.
instance Return the instance of the implementation.

Description

This CreateImpl function creates an instance of the I/O module or the algorithm module. The module descriptor can be a complete module descriptor returned from the QueryImpl function, or a template, in which case the SDK creates an instance of the first I/O or algorithm module matched.

Return Status

PXC_STATUS_NO_ERROR The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE Failed to identify a module that matches the module descriptor, or the requested interface.

Change History

This function was introduced in SDK API 1.0.

Remarks

The application may use the corresponding templates CreateImpl<T> to enable type checking. For example, the following statement creates an instance (into variable y) of interface Y: pxcStatus sts=session->CreateImpl<Y>(&y);
CreateScheduler

Syntax

```c
pxcStatus CreateScheduler(PXCScheduler **instance);
```

Parameters

- `instance`: Return the instance of the SDK scheduler.

Description

This `CreateScheduler` function creates an SDK scheduler instance. It is possible to create multiple schedulers, each with a different priority. The SDK coordinates all scheduling requests based on execution priorities.

Return Status

- `PXC_STATUS_NO_ERROR`: The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

LoadImplFromFile

Syntax

```c
pxcStatus LoadImplFromFile(pxcCHAR *fileName);
```

Parameters

- `filename`: The module shared library name.

Description

This `LoadImplFromFile` function loads a module implementation from a shared library at run time.

Return Status

- `PXC_STATUS_NO_ERROR`: The function returned successfully.

Change History

This function was introduced in SDK API 1.0.
**QueryImpl**

**Syntax**

```c
pxcStatus QueryImpl(ImplDesc *template, pxcU32 idx, ImplDesc *desc);
```

**Parameters**

- **template**
  The module descriptor template; for any field in the `ImplDesc` structure, a zero means matching any values. For bit-OR’ed fields, a none-zero value means matching the corresponding bits.

- **idx**
  The zero-based index value if there are multiple matches.

- **desc**
  The matched complete module descriptor, to be returned.

**Description**

This `QueryImpl` function enumerates available modules according to the search criteria (the module descriptor template `template`). If there are multiple matches, the application needs to increase the `idx` number to retrieve additional matches, until the function returns `PXC_STATUS_ITEM_UNAVAILABLE`.

**Return Status**

- **PXC_STATUS_NO_ERROR**
  The function completed successfully.

- **PXC_STATUS_ITEM_UNAVAILABLE**
  There is no match of modules to the template.

**Change History**

This function was introduced in SDK API 1.0.

---

**QueryVersion**

**Syntax**

```c
pxcStatus QueryVersion(ImplVersion *version);
```

**Parameters**

- **version**
  The SDK API version.

**Description**
This function returns the SDK API version.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function completed successfully.

**Change History**

This function was introduced in SDK API 1.0.

---

### UnloadImplFromFile

**Syntax**

```c
pxcStatus UnloadImplFromFile(pxcCHAR *dllName);
```

**Parameters**

- **dllName** The module shared library name.

**Description**

This UnloadImplFromFile function removes a module implementation. The shared library is unloaded from memory.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function returned successfully.
- **PXC_STATUS_ITEM_UNAVAILABLE** Failed to locate the specified module.

**Change History**

This function was introduced in SDK API 1.0.

---

### PXCSession_Create

**Syntax**

```c
pxcStatus PXCSession_Create(PXCSession **session);
```

**Parameters**

- **session** The PXCSession instance, to be returned.
Description

This function creates an instance of the PXCSession class.

Return Status

PX_C_STATUS_NO_ERROR  The function completed successfully.

Change History

This function was introduced in SDK API 1.0.

PXCSmartArray

The PXCSmartArray template is a utility template to help the application manage arrays of the SDK object instances. See also the PXCSmartPtr template.

The PXCSmartArray<T> template exposes the following operations:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXCSmartArray(int size)</td>
<td>This constructor manages an instance array of type T and size size.</td>
</tr>
<tr>
<td>~PXCSmartArray(void)</td>
<td>The destructor releases the array instances.</td>
</tr>
<tr>
<td>operator **</td>
<td>Return the array instance as an instance pointer array.</td>
</tr>
<tr>
<td></td>
<td>This function is usually used to access individual instances or to</td>
</tr>
<tr>
<td></td>
<td>pass data (e.g., a set of images) from one module to the other:</td>
</tr>
<tr>
<td></td>
<td>Example1:</td>
</tr>
<tr>
<td></td>
<td>PXCSmartArray&lt;PXCImage&gt; images(3);</td>
</tr>
<tr>
<td></td>
<td>PXCSmartSP sp;</td>
</tr>
<tr>
<td></td>
<td>capture_stream-&gt;ReadStreamAsync(&amp;images[0],&amp;sp);</td>
</tr>
<tr>
<td></td>
<td>Example2:</td>
</tr>
<tr>
<td></td>
<td>PXCSmartArray&lt;PXCImage&gt; images(3);</td>
</tr>
<tr>
<td></td>
<td>PXCSmartSP sp;</td>
</tr>
<tr>
<td></td>
<td>util_capture-&gt;ReadStreamAsync(images,&amp;sp);</td>
</tr>
<tr>
<td></td>
<td>face_detector-&gt;ProcessImageAsync(images,face,&amp;sp);</td>
</tr>
<tr>
<td>T* ReleasePtr(int i)</td>
<td>Return the value of array element i and then set the internal pointer to NULL. See the ReleasePtr function in the PXCSmartPtr</td>
</tr>
</tbody>
</table>
template for typical usage.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T** ReleaseRef(int i)</td>
<td>Return the address of array element i after releasing the internal pointer. See the ReleaseRef function in the PXCSmartPtr template for typical usage.</td>
</tr>
<tr>
<td>T** ReleasePtrs()</td>
<td>Return the array instance as an instance pointer array. Internally, all array values are set to zero.</td>
</tr>
<tr>
<td>T** ReleaseRefs()</td>
<td>Return the array instance as an instance pointer array, after releasing all array pointers.</td>
</tr>
<tr>
<td>int QuerySize()</td>
<td>Return the array size.</td>
</tr>
</tbody>
</table>

**PXCSmartPtr**

The PXCSmartPtr template is a utility template to help the application manage the SDK object instances.

The PXCSmartPtr<T> template exposes the following operations:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXCSmart(T* t)</td>
<td>This constructor saves the pointer value to the internal pointer.</td>
</tr>
<tr>
<td>~PXCSmartPtr(void)</td>
<td>The destructor releases the internal pointer if its value is not NULL.</td>
</tr>
<tr>
<td>operator &amp;</td>
<td>Return the address of the internal pointer.</td>
</tr>
<tr>
<td>operator -&gt;</td>
<td>Return the value of the internal pointer.</td>
</tr>
<tr>
<td>operator T*</td>
<td>Return the value of the internal pointer.</td>
</tr>
<tr>
<td>operator =</td>
<td>Assign the value of the internal pointer. The old pointer is released.</td>
</tr>
<tr>
<td>ReleasePtr()</td>
<td>Return the value of the internal pointer and then set the internal pointer to NULL. This function is usually used in recovery from error conditions as follows:</td>
</tr>
</tbody>
</table>

```cpp
PXCIImage *CreateImage(void) {
    PXCSmartPtr<PXCIImage> image2;
    pxcStatus sts=allocator->CreateImage(..., &image2);
    if (sts<PXCI_STATUS_NO_ERROR) return NULL;
    return image2.ReleasePtr();
}
```
## ReleaseRef()

Return the address of the internal pointer after releasing the internal pointer. In the following example, the `ReadStreamAsync` function generates an image instance on each call. The code uses `ReleaseRef()` to automatically release any image instance resulting from previous operations:

```cpp
PXCSmartPtr<PXCImage> image;
for (;;) {
    stream->ReadStreamAsync(..., image.ReleaseRef(), ...);
}
```

## IsValid()

Return if the internal pointer is `NULL` or not.

---

## PXCSmartSP

The `PXCSmartSP` class extends the `PXCSmartPtr` template to help the application manage the SP instances. The `PXCSmartSP` is provided for convenience. It is equivalent to `PXCSmartPtr<PXCScheduler::SyncPoint>`.

## PXCSmartSPArray

The `PXCSmartSPArray` class extends the `PXCSmartArray` template to help the application manage an array of SP instances. The `PXCSmartSPArray` additionally provides the following functions:

```cpp
pxcStatus SynchronizeEx(
    pxcU32 *idx=0,
    pxcU32 timeout=TIMEOUT_INFINITE);
```

Synchronize among the SP instances managed by this template. This function skips any invalid SP values (zero values). See the `SynchronizeEx` function for argument description.
Utility Classes and Function Reference

This section describes SDK utility classes and their member functions. The SDK provides the utility classes in source code under sample/common/include and sample/common/src.

In each interface description, only commonly used status codes are documented. The function may return additional status codes in certain case. See the pxcStatus enumerator for a list of all status codes.

UtilCapture

The UtilCapture interface is a helper interface (on top of the PXCCapture interface) for creating complex pipelines.

The UtilCapture interface provides the following constructors:

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UtilCapture(PXCSession *session);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>session</td>
</tr>
<tr>
<td></td>
<td>The pointer to the PXCSession interface.</td>
</tr>
</tbody>
</table>

The UtilCapture interface provides the following member functions:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryImage</td>
<td>Return the device image of a specific color format.</td>
</tr>
<tr>
<td>QueryDevice</td>
<td>Return the PXCCapture::Device instance.</td>
</tr>
<tr>
<td>QueryVideoStream</td>
<td>Return the PXCCapture::VideoStream instance.</td>
</tr>
<tr>
<td>QueryAudioStream</td>
<td>Return the PXCCapture::AudioStream instance.</td>
</tr>
<tr>
<td>QueryCapture</td>
<td>Return the PXCCapture instance.</td>
</tr>
<tr>
<td>SetFilter</td>
<td>Set filters to limit the input device search.</td>
</tr>
<tr>
<td>LocateStreams</td>
<td>Consolidate all input requests. Find a device that matches them.</td>
</tr>
<tr>
<td>MapImages</td>
<td>Map images from the device view to the module view.</td>
</tr>
</tbody>
</table>
**ReadStreamAsync** | Read a set of samples from the input device.

---

**LocateStreams**

**Syntax**

```
pxcStatus LocateStreams(PXCCapture::VideoStream::DataDesc *inputs);
pxcStatus LocateStreams(PXCCapture::AudioStream::DataDesc *inputs);
pxcStatus LocateStreams(std::vector<PXCCapture::VideoStream::DataDesc*> &vinputs);
pxcStatus LocateStreams(std::vector<PXCCapture::AudioStream::DataDesc*> &ainputs);
pxcStatus LocateStreams(std::vector<PXCCapture::VideoStream::DataDesc*> &vinputs, std::vector<PXCCapture::AudioStream::DataDesc*> &ainputs);
```

**Parameters**

- **inputs**: The input data description of a single audio or video processing module.
- **vinputs**: A vector of input data descriptions from multiple video processing modules.
- **ainputs**: A vector of input data descriptions from multiple audio processing modules.

**Description**

This `LocateStreams` function consolidates all input data descriptions from the processing modules, and locates an input device that can provide the data. The function configures the input device before returning to the application.

If any input data description contains device property descriptions, the function fills the device property values in place in the data descriptions.

For a single processing module, samples in the device view and the module view are equivalent. The application does not need to map the views during streaming.

If there is more than a single processing module, samples in the device view and the module view are not the same. The application must use the `MapImages` function to map the view, or use the `QueryImage` functions to retrieve the image of a specific format.

As there is no implicit audio format conversion, requests from multiple audio processing modules must exactly match each other, or the function is not able to find a device that matches the needs.

**Return Status**
PXC_STATUS_NO_ERROR The function returned successfully.

PXC_STATUS_ITEM_UNAVAILABLE The function failed to locate an input device.

**MapImages**

**Syntax**

```cpp
pxcStatus MapImages(pxci32 module, PXCImage *images_in[], PXCImage *images_out[]);```

**Parameters**

- **module**
  The index of modules, based on the minputs vector of the LocateStreams function.

- **images_in**
  A set of samples from the device view.

- **images_out**
  A set of samples from the module view, to be returned.

**Description**

This MapImages function maps streaming samples from the device view to the module view.

**Return Status**

- **PXC_STATUS_NO_ERROR** The function returned successfully.

**QueryAudioStream**

**Syntax**

```cpp
PXCCapture::AudioStream *QueryAudioStream(void)```

**Description**

This QueryAudioStream function returns the audio stream instance that the LocateStreams function is able to locate.

The function returns NULL if there is no such instance.
### QueryCapture

**Syntax**

```
PXCCapture *QueryCapture(void)
```

**Description**

This `QueryCapture` function returns the `PXCCapture` instance that the `LocateStreams` function is able to locate.

The function returns `NULL` if there is no such instance.

### QueryDevice

**Syntax**

```
PXCCapture::Device *QueryDevice(void)
```

**Description**

This `QueryDevice` function returns the `PXCCapture::Device` instance that the `LocateStreams` function is able to locate.

The function returns `NULL` if there is no such instance.

### QueryImage

**Syntax**

```
PXCImage *QueryImage(PXCImage *images[], PXCImage::ImageType type);
```

**Parameters**

- `images` A set of samples from the device view.
- `type` The desired image type. See the `PXCImage::ImageType` enumerator for definitions.

**Description**

This `QueryImage` function searches the sample of a specific image type.

The function returns `NULL` if there is no such instance.
QueryVideoStream

**Syntax**

```cpp
PXCCapture::VideoStream *QueryVideoStream(int channel, int input=0)
```

**Parameters**

- **Channel**: The zero-based index of streams if a processing module requests more than a single stream.
- **input**: The index of data descriptions corresponding to the `minputs` vector of the `LocateStreams` function.

**Description**

This `QueryVideoStream` function returns the video stream instance that the `LocateStreams` function is able to locate. The function returns `NULL` if there is no such instance.

ReadStreamAsync

**Syntax**

```cpp
pxcStatus ReadStreamAsync(PXCImage *images[], PXCScheduler::SyncPoint **sp);
pxcStatus ReadStreamAsync(PXCAudio **audio, PXCScheduler::SyncPoint **sp);
```

**Description**

This `ReadStreamAsync` function reads audio or image samples from the input device. If more than one stream is requested in the `LocateStreams` function, the function reads all samples at once. The function assumes all streams are generated with the same frame rate.

The output samples are in the device view.

This is an asynchronous function.

**Return Status**

- `PXC_STATUS_NO_ERROR`: The function returned successfully.
SetFilter

Syntax

void SetFilter(PXCSession::ImplDesc &desc);
void SetFilter(PXCIImage::ImageType type, PXCSiz32U &size);
void SetFilter(const pxcCHAR *devname);
void SetFilter(PXCCapture::Device::Property label, pxcF32 value);

Parameters

desc Set a filter for the specified capture module description table.

type/size Set a filter for the specified image resolution and image type.

devname Set a filter for a device whose name contains the specified sub-string. The application must preserve the string buffer until after the LocateStreams function.

label/value Set a filter for a device that responds to the specified device property.

Description

This SetFilter function sets specific search criteria for locating the input device.

UtilCaptureFile

The UtilCaptureFile utility class is a derived class of the UtilCapture utility class for captured data recording and playback.

The UtilCaptureFile interface provides the following constructors:

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UtilCaptureFile(PXCSession *session, pxcCHAR *filename, pxcBool recording);</td>
<td></td>
</tr>
</tbody>
</table>

| session | The pointer to the PXCSession interface. |
| filename | The record file name. |
| recording | If true, the instance is initialized for recording to the specified file. Otherwise, it is initialized for playback of the specified file. |
The **UtilCaptureFile** interface provides the following member functions in addition to what are provided in the **UtilCapture** utility class:

<table>
<thead>
<tr>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SetMask</strong></td>
<td>Set the stream recording mask.</td>
</tr>
<tr>
<td><strong>SetRealtime</strong></td>
<td>Set the realtime mode.</td>
</tr>
<tr>
<td><strong>SetPause</strong></td>
<td>Set the pause playback mode.</td>
</tr>
<tr>
<td><strong>SetPosition</strong></td>
<td>Set the playback position.</td>
</tr>
<tr>
<td><strong>QueryPosition</strong></td>
<td>Query the current playback position.</td>
</tr>
</tbody>
</table>

### QueryPosition

**Syntax**

```c
pxcI32 QueryPosition(void);
```

**Parameters**

None

**Description**

This *QueryPosition* function returns the current playback position in terms of frame numbers.

### SetMask

**Syntax**

```c
void SetMask(PXCImage::ImageType types);
```

**Parameters**

- `types` Bit-OR’ed values to specify the stream types to be recorded. See the *PXCImage::ImageType* enumerator for stream type definitions.

**Description**

This *SetMask* function specifies the set of stream types to be recorded.
SetPause

Syntax

```c
void SetPause(pxcBool pause);
```

Parameters

- `pause` If true, the playback will be in the pause mode by repeating the current frame.

Description

This `SetPlayback` function controls the pause playback mode.

SetPosition

Syntax

```c
void SetPosition(pxCInt32 iframe);
```

Parameters

- `iframe` If positive, the playback will start from the specified frame index.

Description

This `SetPosition` function sets the current playback position in terms of frame numbers.

SetRealtime

Syntax

```c
void SetRealtime(pxcBool realtime);
```

Parameters

- `realtime` If true, the playback will be in real-time mode by delivering samples based on the recorded time stamps. If false, the playback will be as fast as possible.
**Description**

This `SetPlayback` function controls the real-time playback mode.

---

**UtilPipeline**

The **UtilPipeline** utility class provides a simple interface for some (limited) application cases, including finger tracking, voice recognition and face analysis processing from live camera or from a file.

The **UtilPipeline** interface provides the following constructors:

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>UtilPipeline(PXCSession *session=NULL);</code></td>
<td>session is the pointer to the PXCSession interface. The utility class will create an internal session instance if the application does not provide a session instance.</td>
</tr>
<tr>
<td><code>UtilPipeline(PXCSession *session=NULL, const pxCCHAR *filename=NULL, bool recording=false);</code></td>
<td><code>filename/recording</code> If specified, the pipeline operates in the file playback or recording mode.</td>
</tr>
</tbody>
</table>

---

The **UtilPipeline** interface provides the following member functions:

<table>
<thead>
<tr>
<th>Category</th>
<th>Member Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essential Functions</td>
<td><strong>IsAudioFrame</strong></td>
<td>Return whether the acquired frame is an audio frame or not.</td>
</tr>
<tr>
<td></td>
<td><strong>IsImageFrame</strong></td>
<td>Return whether the acquired frame is an image frame or not.</td>
</tr>
<tr>
<td></td>
<td><strong>IsDisconnected</strong></td>
<td>Check the device connection status.</td>
</tr>
<tr>
<td></td>
<td><strong>OnDisconnect</strong></td>
<td>Event that the input device is disconnected.</td>
</tr>
<tr>
<td></td>
<td><strong>OnReconnect</strong></td>
<td>Event that the input device is reconnected.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>OnNewFrame</strong></td>
<td>Event that a new frame is available for processing.</td>
<td></td>
</tr>
<tr>
<td><strong>QuerySession</strong></td>
<td>Retrieve the <strong>PXCSession</strong> instance.</td>
<td></td>
</tr>
<tr>
<td><strong>QueryCapture</strong></td>
<td>Retrieve the <strong>UtilCaptureFile</strong> instance.</td>
<td></td>
</tr>
<tr>
<td><strong>Init</strong></td>
<td>Initialize the pipeline.</td>
<td></td>
</tr>
<tr>
<td><strong>AcquireFrame</strong></td>
<td>Wait until a new frame is available and lock it for processing.</td>
<td></td>
</tr>
<tr>
<td><strong>ReleaseFrame</strong></td>
<td>Release the lock on the current frame.</td>
<td></td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>Close the pipeline.</td>
<td></td>
</tr>
<tr>
<td><strong>LoopFrames</strong></td>
<td>A convenient function that performs Init/AcquireFrame/ReleaseFrame/Close.</td>
<td></td>
</tr>
</tbody>
</table>

### Audio and image data capture

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EnableAudio</strong></td>
<td>Capture a specific type of audio sample</td>
</tr>
<tr>
<td><strong>EnableImage</strong></td>
<td>Capture a specific type of image sample</td>
</tr>
<tr>
<td><strong>OnAudio</strong></td>
<td>Event that the request audio sample is available for processing.</td>
</tr>
<tr>
<td><strong>OnImage</strong></td>
<td>Event that the requested image sample is available for processing.</td>
</tr>
<tr>
<td><strong>QueryImageSize</strong></td>
<td>Retrieve the image resolution (before streaming).</td>
</tr>
<tr>
<td><strong>QueryAudio</strong></td>
<td>Retrieve the audio sample.</td>
</tr>
<tr>
<td><strong>QueryImage</strong></td>
<td>Retrieve the image of a specified image type.</td>
</tr>
</tbody>
</table>

### Face analysis

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EnableFaceLocation</strong></td>
<td>Enable the face detection/tracking module.</td>
</tr>
<tr>
<td><strong>EnableFaceLandmark</strong></td>
<td>Enable the face landmark detection/tracking module.</td>
</tr>
<tr>
<td><strong>OnFaceSetup</strong></td>
<td>Event to fine-tune face module configurations.</td>
</tr>
<tr>
<td><strong>OnFaceLocationSetup</strong></td>
<td>Event to fine-tune face detection configurations.</td>
</tr>
<tr>
<td><strong>OnFaceLandmarkSetup</strong></td>
<td>Event to fine-tune face landmark detection configurations.</td>
</tr>
<tr>
<td><strong>QueryFace</strong></td>
<td>Retrieve the <strong>PXCFaceAnalysis</strong> instance.</td>
</tr>
<tr>
<td><strong>Voice Recognition</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td><strong>PauseFaceLocation</strong></td>
<td>Pause or resume face location detection.</td>
</tr>
<tr>
<td><strong>PauseFaceLandmark</strong></td>
<td>Pause or resume face landmark detection.</td>
</tr>
<tr>
<td><strong>EnableVoiceRecognition</strong></td>
<td>Enable the voice recognition module.</td>
</tr>
<tr>
<td><strong>SetVoiceCommands</strong></td>
<td>Set the command and control mode.</td>
</tr>
<tr>
<td><strong>SetVoiceDictation</strong></td>
<td>Set the dictation mode.</td>
</tr>
<tr>
<td><strong>OnVoiceRecognitionSetup</strong></td>
<td>Modify the voice recognition module configuration parameters.</td>
</tr>
<tr>
<td><strong>OnRecognized</strong></td>
<td>Voice recognition notification.</td>
</tr>
<tr>
<td><strong>OnAlert</strong></td>
<td>Voice recognition alert notification.</td>
</tr>
<tr>
<td><strong>QueryVoiceRecognition</strong></td>
<td>Retrieve the PXCVoiceRecognition instance.</td>
</tr>
<tr>
<td><strong>PauseVoiceRecognition</strong></td>
<td>Pause or resume voice recognition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Finger tracking</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EnableGesture</strong></td>
<td>Enable the hand/finger tracking processing module.</td>
</tr>
<tr>
<td><strong>OnGestureSetup</strong></td>
<td>Event to fine-tune hand/finger tracking configurations.</td>
</tr>
<tr>
<td><strong>OnGesture</strong></td>
<td>Event that a gesture is signaled from the gesture module.</td>
</tr>
<tr>
<td><strong>OnAlert</strong></td>
<td>Event that an alert is signaled from the gesture module.</td>
</tr>
<tr>
<td><strong>QueryGesture</strong></td>
<td>Retrieve the PXCGesture instance.</td>
</tr>
<tr>
<td><strong>PauseGesture</strong></td>
<td>Pause or resume finger tracking.</td>
</tr>
</tbody>
</table>

---

**AcquireFrame**

**Syntax**

```c
bool AcquireFrame(bool wait);
```

**Parameters**

- **wait** A boolean value to indicate whether the function should block until a new frame is available.
Description

This AcquireFrame function waits until a new frame is available and locks it for application processing. The application must call the ReleaseFrame function to release the lock so that the pipeline can process subsequent frames. The application should not invoke lengthy procedures between AcquireFrame and ReleaseFrame.

If wait==false, the AcquireFrame function returns the current status, without blocking the execution.

If the application enables both audio and image processing, the pipeline works in alternating modes between audio and image frames. All audio processing operates with the same sampling rate and all image processing operates with the same frame rate. When the AcquireFrame function returns, the application can use the IsAudioFrame and IsImageFrame functions to determine whether the current frame is an audio frame or an image frame.

Return Status

true The function returned successfully and there is a new frame available.
false There is no frame available for processing at this time, or the function received errors during wait, for example, end of file in the file playback mode.

Close

Syntax

void Close(void);

Parameters

None

Description

This Close function closes the pipeline.

Return Status

None

EnableAudio

Syntax
void EnableAudio(AudioFormat format);
void EnableAudio(AudioFormat format, pxcU32 sampleRate, pxcU32 nchannels);

Parameters

format Specify the audio sample format; see the AudioFormat enumerator for definitions.
sampleRate Optional parameter to specify the audio sample rate in Hz.
nchannels Optional parameter to specify the audio channel number.

Description

This EnableAudio function requests that specific audio samples be part of the pipeline streaming.

Return Status

None

EnableFaceLandmark

Syntax

void EnableFaceLandmark(pxcUID iuid=0);
void EnableFaceLandmark(const pxcCHAR *name);

Parameters

iuid Specify the face processing module by its unique identifier.
name Specify the face processing module by its partial friendly name.

Description

This EnableFaceLandmark function configures the pipeline to enable face landmark detection/tracking processing.

Return Status

None
EnableFaceLocation

Syntax

    void EnableFaceLocation(pxcUID iuid=0);
    void EnableFaceLocation(pxCCHAR *name);

Parameters

    iuid  Specify the face processing module by its unique identifier.
    name  Specify the face processing module by its partial friendly name.

Description

    This EnableFaceLocation function configures the pipeline to enable face location detection/tracking processing.

Return Status

    None

EnableGesture

Syntax

    void EnableGesture(pxcUID iuid=0);
    void EnableGesture(pxCCHAR *name);

Parameters

    iuid  Specify the gesture module by its unique identifier.
    name  Specify the gesture module by its partial friendly name.

Description

    This EnableGesture function configures the pipeline to enable hand/finger tracking and gesture recognition.

Return Status

    None
EnableImage

Syntax

```c
void EnableImage(PXCImage::ColorFormat format);
void EnableImage(PXCImage::ColorFormat format, pxcU32 width, pxcU32 height);
```

Parameters

- **format** Specify the image color format; see the `ColorFormat` enumerator for definitions.
- **width** Optional parameter to specify the image width.
- **height** Optional parameter to specify the image height.

Description

This `EnableImage` function requests that specific image types be part of the pipeline streaming. The application calls this function multiple times for multiple requested image types.

Return Status

None

EnableVoiceRecognition

Syntax

```c
void EnableVoiceRecognition(pxcUID iuid=0);
void EnableVoiceRecognition(pxcCHAR *name);
```

Parameters

- **iuid** The optional module implementation identifier.
- **name** The module friendly name.

Description

This `EnableVoiceRecognition` function enables voice recognition including voice command and control and voice dictation functionalities in the pipeline. The function uses the voice recognition module through the module implementation identifier or the module friendly name.

For voice command and control, the application needs to additionally use the
SetVoiceCommands function to set the active command list before the Init function.

**Return Status**

None

---

**Init**

**Syntax**

```c
bool Init(void);
```

**Description**

This Init function configures the pipeline to be ready for processing.

**Return Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The function returned successfully.</td>
</tr>
<tr>
<td>false</td>
<td>The function failed to initialize the pipeline.</td>
</tr>
</tbody>
</table>

---

**IsAudioFrame**

**Syntax**

```c
bool IsAudioFrame(void);
```

**Description**

This IsAudioFrame function returns whether an audio frame is ready. See the AcquireFrame function for details on the pipeline operating mode.

**Return Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The audio frame is ready.</td>
</tr>
<tr>
<td>false</td>
<td>The audio frame is not ready.</td>
</tr>
</tbody>
</table>

---

**IsDisconnected**

**Syntax**
bool IsDisconnected(void);

**Description**

This `IsDisconnected` function returns the input device connection status.

**Return Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The input device is connected.</td>
</tr>
<tr>
<td>false</td>
<td>The input device is disconnected.</td>
</tr>
</tbody>
</table>

---

### IsImageFrame

**Syntax**

```c
bool IsImageFrame(void);
```

**Description**

This `IsImageFrame` function returns whether an image frame is ready. See the `AcquireFrame` function for details on the pipeline operating mode.

**Return Status**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The image frame is ready.</td>
</tr>
<tr>
<td>false</td>
<td>The image frame is not ready.</td>
</tr>
</tbody>
</table>

---

### LoopFrames

**Syntax**

```c
bool LoopFrames(void);
```

**Description**

This `LoopFrames` function is a convenience function corresponding to the following pseudo code:

```c
if (!Init()) return false;
for (;;) {
    // some device hot-plug code omitted.
    if (!AcquireFrame(true)) break;
    if (!ReleaseFrame()) break;
}
```
Close();
return true;

Return Status

true The function returned successfully.
false The function failed to set up or process the pipeline.

OnAlert

Syntax

void PXCAPI OnAlert(PXCGesture::Alert *data);
void PXCAPI OnAlert(PXCVoiceRecognition::Alert *data);

Parameters

data The alert details as described in the PXCGesture::Alert structure, or in the PXCVoiceRecognition::Alert structure.

Description

The application should overwrite this OnAlert function for processing any alert notification. The default implementation does nothing.

Remarks

The application must use the PXCAPI declaration to properly overwrite this function.

OnAudio

Syntax

void OnAudio(PXCAudio *audio);

Parameters

audio The audio sample instance.

Description

The application should overwrite this OnAudio function for processing any available audio notifications. The default implementation does nothing.
**OnFaceLandmarkSetup**

**Syntax**

```c
void OnFaceLandmarkSetup(PXCFaceAnalysis::Landmark::ProfileInfo *pinfo);
```

**Parameters**

- `pinfo` The face module configurations as described in the `PXCFaceAnalysis::Landmark::ProfileInfo` structure.

**Description**

The application should overwrite this `OnFaceLandmarkSetup` function for fine-tuning the face processing module landmark detection configuration parameters. The default implementation does nothing.

The configuration occurs during the `Init` function call.

---

**OnFaceLocationSetup**

**Syntax**

```c
void OnFaceLocationSetup(PXCFaceAnalysis::Detection::ProfileInfo *pinfo);
```

**Parameters**

- `pinfo` The face module configurations as described in the `PXCFaceAnalysis::Detection::ProfileInfo` structure.

**Description**

The application should overwrite this `OnFaceLocationSetup` function for fine-tuning the face processing module location detection configuration parameters. The default implementation does nothing.

The configuration occurs during the `Init` function call.

---

**OnFaceSetup**

**Syntax**

```c
void OnFaceSetup(PXCFaceAnalysis::ProfileInfo *pinfo);
```

**Parameters**
The face module configurations as described in the 
\texttt{PXCFaceAnalysis::ProfileInfo} structure.

\textbf{Description}

The application should overwrite this \texttt{OnFaceSetup} function for fine-tuning the face processing module configuration parameters. The default implementation does nothing. The configuration occurs during the \texttt{Init} function call.

\subsection*{OnDisconnect}

\textbf{Syntax}

\begin{verbatim}
bool OnDisconnect(void);
\end{verbatim}

\textbf{Description}

The application should overwrite this \texttt{OnDisconnect} function to receive notification when the input device is disconnected. The default implementation sleeps for 10 milliseconds and returns true.

When a device is disconnected, this function will be invoked repeatedly until either the device is reconnected or the function returns \texttt{false}, which terminates the pipeline.

\textbf{Return Status}

\begin{itemize}
  \item \texttt{true} \hspace{1cm} Continue the pipeline processing.
  \item \texttt{false} \hspace{1cm} Stop the pipeline processing.
\end{itemize}

\subsection*{OnGesture}

\textbf{Syntax}

\begin{verbatim}
void PXCAPI OnGesture(PXCGesture::Gesture *data, pxcBool active);
\end{verbatim}

\textbf{Parameters}

\begin{itemize}
  \item \texttt{data} \hspace{1cm} The gesture details as described in the \texttt{PXCGesture::Gesture} structure.
\end{itemize}

\textbf{Description}

The application should overwrite this \texttt{OnGesture} function for processing any gesture notification. The default implementation does nothing.
Remarks

The application must use the PXCAPI declaration to properly overwrite this function.

OnGestureSetup

Syntax

```c
void OnGestureSetup(PXCGesture::ProfileInfo *pinfo);
```

Parameters

- `pinfo` The gesture module configurations as described in the PXCGesture::ProfileInfo structure.

Description

The application should overwrite this OnGestureSetup function for fine-tuning the gesture processing module configuration parameters. The default implementation does nothing.

The configuration occurs during the `Init` function call.

OnImage

Syntax

```c
void OnImage(PXCImage *image);
```

Parameters

- `image` The image instance.

Description

The application should overwrite this OnImage function for processing any available image notifications. The default implementation does nothing.

OnNewFrame

Syntax

```c
bool OnNewFrame(void);
```
**Parameters**

None

**Description**

The application should overwrite this `OnNewFrame` function to perform additional processing when a new frame of data is available. The `LoopFrames` function uses the return value of this function to break out of the pipeline processing.

The default implementation does nothing.

**Return Status**

true  
Continue the pipeline processing.

false  
Stop the pipeline processing.

---

### OnRecognized

**Syntax**

```c
void PXCAPI OnRecognized(PXCVoiceRecognition::Recognition *data);
```

**Parameters**

*data*  
The voice recognition result in the `PXCVoiceRecognition::Recognition` data structure.

**Description**

The application should overwrite the `OnRecognized` function to receive notification on voice recognition. The default implementation does nothing.

**Return Status**

None

**Remarks**

The application must use the `PXCAPI` declaration to properly overwrite this function.

---

### OnReconnect

**Syntax**

---
void OnReconnect(void);

Description

The application should overwrite this OnReconnect function to receive notification when the input device is previously disconnected and then reconnected. The default implementation does nothing.

OnVoiceRecognitionSetup

Syntax

void OnVoiceReconitionSetup(PXCVoiceRecognition::ProfileInfo *pinfo);

Parameters

pinfo The voice recognition module configurations as described in the PXCVoiceRecognition::ProfileInfo structure.

Description

The application should overwrite this OnVoiceRecognitionSetup function to modify any voice recognition module configuration parameters during initialization. The default implementation does nothing.

PauseFaceLandmark

Syntax

void PauseFaceLandmark(bool pause);

Parameters

pause A boolean value to indicate pause or resume.

Description

This PauseFaceLandmark function pauses or resumes face landmark detection processing.

Return Status

None
**PauseFaceLocation**

**Syntax**

```c
void PauseFaceLocation(bool pause);
```

**Parameters**

- **pause**
  A boolean value to indicate pause or resume.

**Description**

This `PauseFaceLocation` function pauses or resumes face location detection processing.

**Return Status**

None

**PauseGesture**

**Syntax**

```c
void PauseGesture(bool pause);
```

**Parameters**

- **pause**
  A boolean value to indicate pause or resume.

**Description**

This `PauseGesture` function pauses or resumes finger tracking processing.

**Return Status**

None

**PauseVoiceRecognition**

**Syntax**

```c
void PauseVoiceRecognition(bool pause);
```

**Parameters**

- **pause**
  A boolean value to indicate pause or resume.
Description

This `PauseVoiceRecognition` function pauses or resumes voice recognition processing.

Return Status

None

QueryAudio

Syntax

```c
PXCAudio *QueryAudio(void);
```

Description

This `QueryAudio` function returns the available audio sample.

Return Status

The `PXCAudio` instance or `NULL` if the requested sample is not available.

QueryCapture

Syntax

```c
PXCCapture *QueryCapture(void);
```

Description

This `QueryCapture` function returns the `PXCCapture` instance.

Return Status

The `PXCCapture` instance or `NULL` if the capture module is not initialized.

QueryFace

Syntax

```c
PXCFaceAnalysis *QueryFace(void);
```

Description
This QueryFace function returns the PXCFaceAnalysis instance.

**Return Status**

The PXCFaceAnalysis instance or NULL if the face module is not initialized.

---

**QueryGesture**

**Syntax**

```c
PXCGesture *QueryGesture(void);
```

**Description**

This QueryGesture function returns the PXCGesture instance.

**Return Status**

The PXCGesture instance or NULL if the gesture module is not initialized.

---

**QueryImage**

**Syntax**

```c
PXCImage *QueryImage(PXCImage::ImageType type);
```

**Description**

This QueryImage function retrieves a specific type of image sample from the current available samples.

**Return Status**

The PXCImage instance or NULL if the requested image is not available.

---

**QueryImageSize**

**Syntax**

```c
bool QueryImageSize(PXCImage::ImageType type, pxcU32 &width, pxcU32 &height);
```

**Description**
This `QueryImageSize` function returns the resolution of the specified image types. The resolution information is available after the pipeline is initialized.

**Return Status**

- **true**: The function returned successfully.
- **false**: The function failed to retrieve the resolution information.

---

### QuerySession

**Syntax**

```c
PXCSession *QuerySession(void);
```

**Description**

This `QuerySession` function returns the `PXCSession` instance.

**Return Status**

The `PXCSession` instance.

---

### QueryVoiceRecognition

**Syntax**

```c
PXCVoicRecognition *QueryVoiceRecognition(void);
```

**Description**

This `QueryVoiceRecognition` function returns the `PXCVoicRecognition` instance.

**Return Status**

The `PXCVoicRecognition` instance.

---

### ReleaseFrame

**Syntax**

```c
bool ReleaseFrame(void);
```
**ReleaseFrame**

This function releases the lock on the current frame. The pipeline continues to process subsequent frames.

**Return Status**

- **true**: The function returned successfully.
- **false**: The function failed to release the lock.

---

**SetVoiceCommands**

**Syntax**

```cpp
void SetVoiceCommands(std::vector<std::wstring> &cmds);
```

**Parameters**

- **cmds**: The list of commands to be set.

**Description**

This function enables the command and control mode by specifying a list of commands.

**Return Status**

- **None**: The function executed successfully.

---

**SetVoiceDictation**

**Syntax**

```cpp
void SetVoiceDictation(void);
```

**Parameters**

- **None**: The function executed successfully.

**Description**

This function enables the voice dictation mode.

**Return Status**

- **None**: The function executed successfully.
None
In the following structure references, all reserved fields must be zero.

**PXCAudio::AudioData**

**Definition**

```c
struct AudioData {
    AudioFormat format;
    SurfaceType type;
    pxcU32 dataSize;
    pxcU32 reserved;
    union {
        pxcBYTE *dataPtr;
        pxcHDL surfaceHandle;
    }
};
```

**Description**

The AudioData structure describes audio storage details.

**Members**

- **format**: The audio sample format; see the AudioFormat enumerator for definitions.
- **type**: The audio storage format; see the SurfaceType enumerator for definitions.
- **dataSize**: Valid data size in number of audio samples.
- **dataPtr**: The data buffer pointer.
- **surfaceHandle**: The OS surface handle

**Change History**

This structure is introduced in SDK API 1.0.
PXCAudio::AudioInfo

Definition

```c
struct AudioInfo {
    pxcU32 bufferSize;
    AudioFormat format;
    pxcU32 sampleRate;
    pxcU32 nchannels;
    ChannelMask channelMask;
};
```

Description

The AudioData structure describes audio storage details.

Members

- bufferSize: Maximum buffer size in number of audio samples.
- format: The audio storage format; see the AudioFormat enumerator for definitions.
- sampleRate: Samples per second.
- nchannels: Number of audio channels.
- channelMask: The channel layout; see the ChannelMask enumerator for definitions.

Change History

This structure is introduced in SDK API 1.0.

PXCCapture::AudioStream::DataDesc

Definition

```c
struct DataDesc {
    PXCAudio::AudioInfo info;
    PXCAudio::AudioOption options;
};
```
Description

The DeviceCap structure describes the audio stream I/O data needs for a processing module. Unused fields must be zero’ed.

Members

- audioInfo: The audio properties in the PXCAudio::AudioInfo structure.
- audioOptions: The audio options in the PXCAudio::AudioOption enumerator.

Change History

This structure is introduced in SDK API 1.0.

---

PXCCapture::AudioStream::ProfileInfo

Definition

```c
struct ProfileInfo {
    PXCAudio::AudioInfo audioInfo;
    PXCAudio::AudioOption audioOptions;
};
```

Description

The ProfileInfo structure describes audio configuration parameters.

Members

- audioInfo: The audio properties in the PXCAudio::AudioInfo structure.
- audioOptions: The audio options in the PXCAudio::AudioOption enumerator.

Change History

This structure is introduced in SDK API 1.0.

---

PXCCapture::DeviceInfo

Definition
struct DeviceInfo {
    pxcCHAR name[256];
    pxcCHAR did[256];
    pxcU32 didx;
    pxCEnum options;
    pxcU32 reserved;
};

Description
The **DeviceInfo** structure provides the capture device information.

Members
- didx The device index.
- name The NULL-terminated, user-friendly device name string.
- did The NULL-terminated, unique string to identify the capture device.
- options Reserved; must be zero.

Change History
This structure is introduced in SDK API 1.0.

**PXCCapture::Device::DeviceCap**

Definition

```c
struct DeviceCap {
    Property label;
    pxcF32 value;
};
```

Description
The **DeviceCap** structure describes a device property and its value.

Members
- label The device property label. See the **Property** enumerator for definitions. If this value is negative, the property (specified by its absolute value) is
an optional property.

value The property value.

Change History

This structure is introduced in SDK API 1.0.

PXCCapture::Device::StreamInfo

Definition

```c
struct StreamInfo {
    pxcU32 sidx;
    pxcUID cuid;
    PXCIImage::ImageType imageType;
    pxcU32 reserved;
};
```

Description

The StreamInfo structure describes the stream information of a capture device.

Members

sidx The stream index.

cuid The stream interface identifier. For video streams, this is PXCCapture::VideoStream::CUID. For audio streams, this is PXCCapture::AudioStream::CUID.

imageType The image type for video streams. See the PXCIImage::ImageType enumerator for definitions.

Change History

This structure is introduced in SDK API 1.0.

PXCCapture::VideoStream::DataDesc

Definition
struct DataDesc {
    StreamDesc streams[STREAM_LIMIT];
    Device::DeviceCap devCaps[Device::DEVCAP_LIMIT];
};

Description

The DeviceCap structure describes the video stream I/O data needs for a processing module. Unused fields must be zero’ed.

Members

streams  Description of all streams needed by the processing module. See the StreamDesc structure for stream description definitions.

devCaps  Description of additional device properties required by the processing module. See the DeviceCap structure for device property definitions.

Change History

This structure is introduced in SDK API 1.0.

Definition

struct StreamDesc {
    PXCSIZEU32 sizeMin;
    PXCSIZEU32 sizeMax;
    PXCI::ColorFormat format;
    PXCI::ImageOption options;
};

Description

The StreamDesc structure describes the stream characteristics of a module’s I/O data requirements. Unused fields should be zero’ed.

Members

sizeMin  If not zero, specify the minimum resolution.

sizeMax  If not zero, specify the maximum resolution.
format  The image color format. See the PXCImage::ColorFormat enumerator for definitions. This field must be specified.

options  Image color format options. See the PXCImage::ImageOption enumerator for definitions.

Change History
This structure is introduced in SDK API 1.0.

PXCCapture::VideoStream::ProfileInfo

Definition

```c
struct ProfileInfo {
  PXCImage::ImageInfo imageInfo;
  PXCRatioU32 frameRateMin;
  PXCRatioU32 frameRateMax;
  PXCImage::ImageOption imageOptions
};
```

Description
The ProfileInfo structure describes video stream profile configuration.

Members

- **imageInfo**: The image properties in the PXCImage::ImageInfo structure.
- **frameRateMin**, **frameRateMax**: The range of supported frame rates. When used in the SetProfile function, the application should set both values to the requested frame rate.
- **imageOptions**: The image creation options in the PXCImage::ImageOption enumerator.

Change History
This structure is introduced in SDK API 1.0.
PXCIImage::ImageData

Definition

```c
struct ImageData {
    ColorFormat          format;
    SurfaceType          type;
    pxcU32               reserved;
    union {
        struct {
            pxcU32      pitches[4];
            pxcBYTE     *planes[4];
        }
        pxcHDL surfaceHandle;
    }
};
```

Description

The `ImageData` structure describes image storage details. The SDK uses color planes and pitches to specify images stored in system memory buffers, and uses the OS surface handle to refer to images stored in the OS surface.

Members

- **format**: The color format of image buffer. See the `ColorFormat` enumerator for definitions.
- **type**: The image storage format; see the `SurfaceType` enumerator for definitions.
- **planes**: Image color plane buffers. Compacted or compressed images use `planes[0]` to point to a single continuous buffer.
- **pitches**: Pitches of the color plane buffers.
- **surfaceHandle**: The OS surface handle

Change History

This structure is introduced in SDK API 1.0.
PXCImage::ImageInfo

Definition

```c
struct ImageInfo {
    pxcU32 width;
    pxcU32 height;
    ColorFormat format;
    pxcU32 reserved;
};
```

Description

The `ImageInfo` structure defines the essential properties of an image storage.

Members

width The image width in pixels.

height The image height in pixels.

format The image color format. See the `ColorFormat` enumerator for definitions.

Change History

This structure is introduced in SDK API 1.0.

PXCSession::ImplDesc

Definition

```c
struct ImplDesc {
    ImplGroup group;
    ImplSubgroup subgroup;
    pxCUID algorithm;
    pxCUID iuid;
    ImplVersion version;
    PXCAccelerator::AccelType acceleration;
    pxcU32 merit;
};
```
pxcU32 vendor;
pxcUID cuids[4];
pxcCHAR friendlyName[256];
pxcU32 reserved[12];
};

**Description**

The **ImplDesc** structure describes a module implementation. The application can use this structure as a template to search for modules. In this case, zero in a structure field will match any values. A bit-pattern in a bit-OR'ed field will match any bit patterns that are a superset.

**Members**

- **group** The module group; see the [ImplGroup](#) enumerator for details.
- **subgroup** The module subgroup; see the [ImplSubgroup](#) enumerator for details.
- **algorithm** The algorithm identifier in the module implementation; see the module interface for the algorithm enumerations.
- **iuid** The module implementation identifier;
- **version** The module implementation version; see the [ImplVersion](#) structure for details.
- **acceleration** List supported acceleration frameworks. See the [AccelType](#) enumerator for details.
- **merit** The preference value. Modules with a higher merit value take precedence in the module inquiry ([QueryImpl](#)) and creation ([CreateImpl](#)).
- **vendor** The vendor identifier. The Intel vendor identifier is 0x8086. Other vendors that implement the modules may define their own identifiers.
- **cuids** The first four exposed interface identifiers.
- **friendlyName** The NULL-terminated friendly name of the module.

**Change History**

This structure is introduced in SDK API 1.0.

**Remarks**

The **group** and **subgroup** values uniquely identify an SDK interface. The **iuid** value uniquely identifies a module implementation that implements a particular SDK interface. Thus within an SDK session, the following combinations uniquely identify a module
implementation:

(1) group, subgroup, and iuid
(2) cuid and iuid

Here cuid refers to the identifier of the primary interface that the module implements.

**PXCSession::ImplVersion**

**Definition**

```c
struct ImplVersion {
    mfxU16 major;
    mfxU16 minor;
};
```

**Description**

The ImplVersion structure defines the SDK API or module implementation version numbers.

**Members**

- **major**
  The major version number.

- **minor**
  The minor version number.

**Change History**

This structure is introduced in SDK API 1.0.
## Enumerator Reference

### PXCAccelerator::AccelType

**Description**

The `AccelType` enumerator uses bit-OR’ed values to represent the acceleration frameworks.

**Name/Description**

- **ACCEL_TYPE_ANY**: Any acceleration.
- **ACCEL_TYPE_CPU**: CPU instruction-level acceleration.
- **ACCEL_TYPE_GPU_VIA_DX9**: Acceleration through the Microsoft Direct3D* 9 framework.

**Change History**

This enumerator is introduced in SDK API 1.0.

### PXCAccelerator::HandleType

**Description**

The `HandleType` enumerator itemizes OS-specific acceleration framework handles.

**Name/Description**

- **HANDLE_TYPE_DX9_DEVICE**: The Microsoft Direct3D IDirect3DManager9 instance.

**Change History**

This enumerator is introduced in SDK API 1.0.

### PXCAudio::Access

**Description**
The Access enumerator itemizes the surface/buffer access modes.

**Name/Description**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_READ</td>
<td>Read access.</td>
</tr>
<tr>
<td>ACCESS_WRITE</td>
<td>Write access.</td>
</tr>
<tr>
<td>ACCESS_READ_WRITE</td>
<td>Read and write access.</td>
</tr>
</tbody>
</table>

**Change History**

This enumerator is introduced in SDK API 1.0.

---

**PXCAudio::AudioFormat**

**Description**

The AudioFormat enumerator itemizes supported audio formats.

**Name/Description**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIO_FORMAT_PCM</td>
<td>The 16-bit linear PCM format.</td>
</tr>
<tr>
<td>AUDIO_FORMAT_IEEE_FLOAT</td>
<td>The 32-bit floating point format.</td>
</tr>
</tbody>
</table>

**Change History**

This enumerator is introduced in SDK API 1.0.

**Remark**

The application can obtain the sample size in bits by masking the format value with AUDIO_FORMAT_SIZE_MASK.

---

**PXCAudio::AudioOption**

**Description**

The AudioOption enumerator itemizes supported audio options. Currently there is no defined audio option.

**Name/Description**
Change History

This enumerator is introduced in SDK API 1.0.

PXCAudio::ChannelMask

Description

The ChannelMask enumerator uses bit-OR’ed values to describe the audio channel mixes/layout.

Name/Description

<table>
<thead>
<tr>
<th>Name/Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANNEL_MASK_FRONT_LEFT</td>
<td>The front left channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_FRONT_RIGHT</td>
<td>The front right channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_FRONT_CENTER</td>
<td>The front center channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_LOW_FREQUENCY</td>
<td>The low frequency channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_BACK_LEFT</td>
<td>The back left channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_BACK_RIGHT</td>
<td>The back right channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_SIDE_LEFT</td>
<td>The side left channel.</td>
</tr>
<tr>
<td>CHANNEL_MASK_SIDE_RIGHT</td>
<td>The side right channel.</td>
</tr>
</tbody>
</table>

Change History

This enumerator is introduced in SDK API 1.0.

PXCAudio::SurfaceType

Description

The SurfaceType enumerator uses bit-OR’ed values to represent supported surface types.

Name/Description

<table>
<thead>
<tr>
<th>Name/Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURFACE_TYPE_SYSTEM_MEMORY</td>
<td>The memory buffer object.</td>
</tr>
</tbody>
</table>
**SURFACE_TYPE_DX9_SURFACE** The Microsoft DirectX 9 surface type.

**Change History**

This enumerator is introduced in SDK API 1.0.

---

## PXCCapture::Device::Property

**Description**

The `Property` enumerator itemizes supported device properties (configuration parameters.)

**Name/Description**

**Single Value Properties**

- **PROPERTY_COLOR_EXPOSURE** The color camera exposure, in log base 2 seconds.
- **PROPERTY_COLOR_BRIGHTNESS** The color camera brightness from -10,000 (pure black) to 10,000 (pure white).
- **PROPERTY_COLOR_CONTRAST** The color camera contrast, from 0 to 10,000.
- **PROPERTY_COLOR_SATURATION** The color camera saturation, from 0 to 10,000.
- **PROPERTY_COLOR_HUE** The color camera hue, from -180,000 to 180,000 (representing -180 to 180 degrees.)
- **PROPERTY_COLOR_GAMMA** The color camera gamma, from 1 to 500.
- **PROPERTY_COLOR_WHITE_BALANCE** The color camera balance, as a color temperature in degrees Kelvin.
- **PROPERTY_COLOR_SHARPNESS** The color camera sharpness, from 0 to 100.
- **PROPERTY_COLOR_GAIN** The color camera gain adjustment, with negative values darker, positive values brighter, and zero as normal.
- **PROPERTY_COLOR_BACK_LIGHT_COMPENSATION** The color camera back light compensation, with 1 to turn on and 0 to turn off.
- **PROPERTY_DEPTH_LOW_CONFIDENCE_VALUE** The special depth map value to indicate that the corresponding depth map pixel is of low-confidence. The following are possible reasons for low-confidence:
Objects are too close to the camera.
Objects are too far away from the camera.
Objects have low infrared reflectivity.

**PROPERTY_DEPTH_SATURATION_VALUE**
The special depth map value to indicate that the corresponding depth map pixel is saturated. The following are possible reasons for saturation:
- Objects are too close to the camera.
- Objects have a high infrared reflectivity.
- An external infrared source saturates the camera.

**PROPERTY_DEPTH_CONFIDENCE_THRESHOLD**
The confidence threshold that is used to floor the depth map values. The range is from 1 to 32767.

**PROPERTY_DEPTH_SMOOTHING**
If the value is not zero, enable depth value smoothing.

**PROPERTY_AUDIO_MIX_LEVEL**
The microphone recording level, from 0 to 1.

**Two-value Properties**

**PROPERTY_COLOR_FIELD_OF_VIEW**
The color sensor horizontal and vertical field of view parameters, in degrees, as defined in the PXCPPointF32 structure.

**PROPERTY_COLOR_SENSOR_RANGE**
The color sensor sensing distance parameters, in millimeters, as defined in the PXCRangeF32 structure.

**PROPERTY_COLOR_FOCAL_LENGTH**
The color sensor focal length in pixels along the x and y axes, as defined in the PXCPPointF32 structure. The parameters vary with the color stream resolution setting.

**PROPERTY_COLOR_PRINCIPAL_POINT**
The color sensor principal point in pixels along the x and y axes, as defined in the PXCPPointF32 structure. The parameters vary with the color stream resolution setting.

**PROPERTY_DEPTH_FIELD_OF_VIEW**
The depth sensor horizontal and vertical field of view parameters, in degrees, as defined in the PXCPPointF32 structure.

**PROPERTY_DEPTH_SENSOR_RANGE**
The depth-sensor, sensing distance parameters, in millimeters, as defined in the PXCRangeF32 structure.

**PROPERTY_DEPTH_FOCAL_LENGTH**
The depth sensor focal length in pixels along the x and y axes, as defined in the PXCPPointF32 structure.
structure. The parameters vary with the depth stream resolution setting.

PROPERTY_DEPTH_PRINCIPAL_POINTS

The depth sensor principal point in pixels along the x and y axes, as defined in the PXCPointF32 structure. The parameters vary with the depth stream resolution setting.

Miscellaneous Properties

PROPERTY_ACCELEROMETER_READING

The accelerometer reading, in world coordinates and in the g unit, in the PXCPoint3DF32 structure.

PROPERTY_PROJECTION_SERIALIZABLE

The metadata identifier of the PXCPProjection implementation, attached to the session metadata.

Customized Properties

PROPERTY_CUSTOMIZED

Customized controls. Any customized controls should have a label value bigger than LABEL_CUSTOMIZED.

Change History

This enumerator is introduced in SDK API 1.0.

PXCImage::Access

Description

The Access enumerator itemizes the surface/buffer access modes.

Name/Description

ACCESS_READ

Read access.

ACCESS_WRITE

Write access.

ACCESS_READ_WRITE

Read and write access.

Change History

This enumerator is introduced in SDK API 1.0.
PXCImage::ColorFormat

Description

The ColorFormat enumerator itemizes supported image color formats.

Name/Description

Color images (IMAGE_TYPE_COLOR)

COLOR_FORMAT_YUY2 The YUY2 color format.
COLOR_FORMAT_NV12 The NV12 color format.
COLOR_FORMAT_RGB32 The 32-bit RGB32 color format. On little endian machines, the memory layout is BGRA.
COLOR_FORMAT_RGB24 The 24-bit RGB24 color format. On little endian machines, the memory layout is BGR.
COLOR_FORMAT_GRAY The 8-bit gray format.

Depth images: (IMAGE_TYPE_DEPTH)

COLOR_FORMAT_DEPTH The depth map, which contains three planes, unless otherwise specified in ImageOption. The first plane contains a depth map; the second plane contains the corresponding confidence map; and the third plane contains the corresponding UV map.
COLOR_FORMAT_VERTICES The fixed-point vertices map, which contains three planes, unless otherwise specified in ImageOption. The first plane contains the vertices in fixed-point integers; the second plane contains the corresponding confidence map; and the third plane contains the corresponding UV map.

Change History

This enumerator is introduced in SDK API 1.0.

Remarks

The SDK defines color formats in terms of image types. To obtain the image type from an image color format, the application can use the formula (color_format&IMAGE_TYPE_MASK). See the ImageType enumerator for details.

For the depth map, each pixel is a 16-bit integer. The value indicates the distance from an object to the camera XY plane or the Cartesian depth, in millimeters. The depth map
values are thresholded if they are below certain a confidence threshold or saturated. See the \texttt{PROPERTY_DEPTH_LOW_CONFIDENCE_VALUE} and the \texttt{PROPERTY_DEPTH_SATURATED_VALUE} device properties for details about thresholding and saturation.

For confidence maps, each pixel is a 16-bit integer. The value is the confidence value from the corresponding depth pixel. Higher is better.

For fixed-point vertices, each pixel contains three 16-bit integers, representing the world coordinates \((x, y, \text{and depth value})\).

For UV maps, each pixel contains two 32-bit floating point values in the range of 0-1, representing the mapped depth coordinates to the color coordinates.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{PXCImage::ImageOption} & \\
\hline
\textbf{Description} & The \texttt{ImageOption} enumerator itemizes supported image options. \\
\hline
\textbf{Name/Description} & \\
\hline
\texttt{IMAGE_OPTION_NO_UV_MAP} & Create a depth image without the UV map. \\
\texttt{IMAGE_OPTION_NO_CONFIDENCE_MAP} & Create a depth image without the confidence map. \\
\hline
\end{tabular}
\caption{PXCImage::ImageOption Enumerations}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{PXCImage::ImageType} & \\
\hline
\textbf{Description} & The \texttt{ImageType} enumerator itemizes supported image types. \\
\hline
\textbf{Name/Description} & \\
\hline
\texttt{IMAGE_TYPE_COLOR} & The image is a color image. \\
\texttt{IMAGE_TYPE_DEPTH} & The image is a depth image. \\
\hline
\end{tabular}
\caption{PXCImage::ImageType Enumerations}
\end{table}
Change History

This enumerator is introduced in SDK API 1.0.

Remarks

To obtain the image type from an image color format, the application can use the formula (color_format&IMAGE_TYPE_MASK). See the ColorFormat enumerator for details.

PXCImage::SurfaceType

Description

The SurfaceType enumerator uses bit-OR’ed values to represent supported surface types.

Name/Description

<table>
<thead>
<tr>
<th>Name/Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURFACE_TYPE_SYSTEM_MEMORY</td>
<td>The memory buffer object.</td>
</tr>
<tr>
<td>SURFACE_TYPE_DX9_SURFACE</td>
<td>The Microsoft Direct3D 9 surface type.</td>
</tr>
<tr>
<td>SURFACE_TYPE_DX11_SURFACE</td>
<td>The Microsoft Direct3D 11 surface type.</td>
</tr>
<tr>
<td>SURFACE_TYPE_OCL_MEMOBJ</td>
<td>The OpenCL* memory object type.</td>
</tr>
</tbody>
</table>

Change History

This enumerator is introduced in SDK API 1.0.

PXCPowerState::State

Description

The State enumerator itemizes module power state.

Name/Description

<table>
<thead>
<tr>
<th>Name/Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_ADAPTIVE</td>
<td>The module automatically controls its power state.</td>
</tr>
<tr>
<td>STATE_C0</td>
<td>The module works normally. All features are available.</td>
</tr>
</tbody>
</table>
STATE_C1
The module works in the power-saving mode, reducing features or simplifying algorithms to save power.

STATE_C2
The module works in a further power-constrained mode, compared to STATE_C1.

STATE_C3
The module is in the idle mode. Features, except certain trivial ones, are not available.

Change History
This enumerator is introduced in SDK API 1.0.

Remarks
Regardless of power states, the application can expect the SDK modules to behave the same from the SDK API point of view. For example, the application can expect an asynchronous function to return immediately, even when the module is in the idle state. The synchronization of the computation result could take longer, or the quality of the computation could be different due to the algorithms used in different power states.

PXCScheduler::Priority

Description
The Priority enumerator itemizes the task scheduling priorities.

Name/Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY_NORMAL</td>
<td>The scheduler works in the normal priority mode. This is the default.</td>
</tr>
<tr>
<td>PRIORITY_LOW</td>
<td>The scheduler works in the low priority mode.</td>
</tr>
<tr>
<td>PRIORITY_HIGH</td>
<td>The scheduler works in the high priority mode.</td>
</tr>
</tbody>
</table>

Change History
This enumerator is introduced in SDK API 1.0.
PXCSession::ImplGroup

Description

The ImplGroup enumerator itemizes algorithm implementations into major groups. It can also be used as bit-OR’ed values to represent groups of algorithms.

Name/Description

<table>
<thead>
<tr>
<th>Name/Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPL_GROUP_ANY</td>
<td>Any algorithm.</td>
</tr>
<tr>
<td>IMPL_GROUP_CORE</td>
<td>The core framework algorithms.</td>
</tr>
<tr>
<td>IMPL_GROUP_SENSOR</td>
<td>The sensor algorithms.</td>
</tr>
<tr>
<td>IMPL_GROUP_OBJECT_RECOGNITION</td>
<td>The object recognition algorithms.</td>
</tr>
<tr>
<td>IMPL_GROUP_SPEECH_RECOGNITION</td>
<td>The speech recognition algorithms.</td>
</tr>
<tr>
<td>IMPL_GROUP_USER</td>
<td>User-defined algorithms.</td>
</tr>
</tbody>
</table>

Change History

This enumerator is introduced in SDK API 1.0.

PXCSession::ImplSubgroup

Description

The ImplSubgroup enumerator itemizes algorithm implementations into minor groups. It can also be used as bit-OR’ed values to represent groups of algorithms.

Name/Description

<table>
<thead>
<tr>
<th>Name/Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPL_SUBGROUP_ANY</td>
<td>Any algorithm.</td>
</tr>
<tr>
<td>IMPL_SUBGROUP_AUDIO_CAPTURE</td>
<td>Audio capture algorithm.</td>
</tr>
<tr>
<td>IMPL_SUBGROUP_VIDEO_CAPTURE</td>
<td>Video capture algorithm.</td>
</tr>
</tbody>
</table>

Sensor group modules

Object recognition modules
IMPL_SUBGROUP_FACE_ANALYSIS

Face analysis algorithm including detection/tracking, landmark detection, recognition, and attribute detections.

IMPL_SUBGROUP_GESTURE_RECOGNITION

Gesture recognition algorithm.

Speech recognition modules

IMPL_SUBGROUP_VOICE_RECOGNITION

Voice recognition module including dictation and command and control.

IMPL_SUBGROUP_VOICE_SYNTHESIS

Voice synthesis module.

Change History

This enumerator is introduced in SDK API 1.0.

Remarks

See additional subgroup definitions in each module SDK interface definition.

description

The pxcStatus enumerator itemizes status codes returned by SDK functions.

Name/Description

Successful operation

PXC_STATUS_NO_ERROR

The operation completed successfully.

Programming errors

PXC_STATUS_HANDLE_INVALID

The session/module instance or pointer is invalid.

PXC_STATUS_ALLOC_FAILED

Failed to allocate memory, or created an instance of a module or resource.

Configuration-related errors or warnings

PXC_STATUS_FEATURE_UNSUPPORTED

The requested feature is not available or implemented.

PXC_STATUS_PARAM_UNSUPPORTED

There are invalid/unsupported parameters in the
configuration.

PXC_STATUS_ITEM_UNAVAILABLE The item could not be found, or end of stream.

Asynchronous operation-related errors or warnings

PXC_STATUS_EXEC_ABORTED The asynchronous pipeline operation is aborted in upstream components. Check the status return code of upstream components for exactly where the error occurred.

PXC_STATUS_EXEC_INPROGRESS The asynchronous operation is in progress.

PXC_STATUS_EXEC_TIMEOUT The synchronization function timed out. The asynchronous operation is still in progress.

Hardware device-related errors or warnings

PXC_STATUS_DEVICE_FAILED Unexpected device failure due to device malfunctioning, failed to allocate acceleration resources, etc. The application must close the module instance and recreate it.

PXC_STATUS_DEVICE_LOST Unexpected device failure due to the loss of the device such as system sleep/hibernation, or the peripheral device was unplugged. The application can choose to abort or wait for the device to reconnect. In the latter case, the device operation will resume.

PXC_STATUS_DEVICE_BUSY The application does not have control of the device thus cannot change the device configuration. The application can only use the existing configuration of the device.

Misc. errors or warnings

PXC_STATUS_TIME_GAP There is a gap in time stamps.

PXC_STATUS_PARAM_INPLACE The algorithm is already configured with the same parameters.

Change History

This enumerator is introduced in SDK API 1.0.
## Appendices

### Appendix A: Data Type Definitions

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pxcBool</code></td>
<td>Boolean (32-bit integer)</td>
<td><code>pxcI32</code></td>
<td>32-bit integer</td>
</tr>
<tr>
<td><code>pxcBYTE</code></td>
<td>8-bit unsigned char</td>
<td><code>pxcOption</code></td>
<td>32-bit integer</td>
</tr>
<tr>
<td><code>pxcCHAR</code></td>
<td>16-bit UNICODE</td>
<td><code>pxcU16</code></td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td><code>pxcEnum</code></td>
<td>32-bit integer</td>
<td><code>pxcU32</code></td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td><code>pxcF32</code></td>
<td>32-bit floating-point</td>
<td><code>pxcU64</code></td>
<td>64-bit unsigned integer</td>
</tr>
<tr>
<td><code>pxcF64</code></td>
<td>64-bit floating-point</td>
<td><code>pxcU8</code></td>
<td>8-bit unsigned integer</td>
</tr>
<tr>
<td><code>pxcHDL</code></td>
<td>Native pointer</td>
<td><code>pxcUID</code></td>
<td>32-bit integer</td>
</tr>
<tr>
<td><code>pxcI16</code></td>
<td>16-bit integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>pxcPoint3DF32</code></td>
<td>3D point in floating-point coordinates: { <code>pxcF32</code> x, y, z; }.</td>
<td><code>pxcRatioF32</code></td>
<td>Ratio in floating-point numbers: { <code>pxcF32</code> denominator, numerator; }</td>
</tr>
<tr>
<td><code>pxcPointF32</code></td>
<td>2D point in floating-point coordinates: { <code>pxcF32</code> x, y; }</td>
<td><code>pxcRatioU32</code></td>
<td>Ratio in unsigned integers: { <code>pxcU32</code> denominator, numerator; }</td>
</tr>
<tr>
<td><code>pxcPointU32</code></td>
<td>2D point in unsigned integer</td>
<td><code>pxcRectU32</code></td>
<td>Rectangle in unsigned integer</td>
</tr>
<tr>
<td><code>pxcRectU32</code></td>
<td>rectangle in unsigned integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
coordinates: \{ \texttt{pxcU32} x, y; \} 

integer coordinates: \{ \texttt{pxcU32} x, y, w, h; \}

<table>
<thead>
<tr>
<th>\texttt{pxcRangeF32}</th>
<th>Range in floating-point numbers: { \texttt{pxcF32} min, max; }</th>
<th>\texttt{pxcSizeF32}</th>
<th>Size in floating-point numbers: { \texttt{pxcF32} width, height; }</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{pxcRangeI32}</td>
<td>Range in integers: { \texttt{pxcI32} min, max; }</td>
<td>\texttt{pxcSizeU32}</td>
<td>Size in unsigned integers: { \texttt{pxcU32} width, height; }</td>
</tr>
<tr>
<td>\texttt{pxcRangeU32}</td>
<td>Range in unsigned integers: { \texttt{pxcU32} min, max; }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Appendix B: C# Language Support**

The C# wrapper source code is located under \texttt{framework/CSharp/libpxcclr}. The compiled DLL files \texttt{libpxcclr.dll} are under the \texttt{bin/win32} and \texttt{bin/x64} directories. The C# wrapper is a straight porting from the C++ interface. Refer to the C++ functions in this manual and additional SDK manuals for the function definitions. The differences are as follows:

- C# classes and definitions use the \texttt{PXCM} or \texttt{pxcm} prefix. For example, \texttt{PXCSession} becomes \texttt{PXCMSession}. \texttt{pxcStatus} becomes \texttt{pxcmStatus}.
- Unlike in C++, where the application can directly use enumeration definitions, in C#, the application must use the enumeration name as a namespace of the enumeration definitions. For example, \texttt{PXC\_STATUS\_NO\_ERROR} becomes \texttt{pxcmStatus.PXCM\_STATUS\_NO\_ERROR}.
- Use the \texttt{PXCMSession\_CreateSession} function to create a SDK session. See \texttt{PXCMSession\_Create} function for details.
- All SDK classes implement the \texttt{IDispose} interface. The application must explicitly call the \texttt{Dispose} function to clean up any resources after use.
- If an SDK module exposes multiple interfaces, the application should use the \texttt{DynamicCast} function to cast the module instance to different interfaces.
- The C# wrapper ports certain simple query and set functions as properties. For example, the \texttt{PXCSession::QueryVersion} becomes a property of \texttt{PXCMSession}.
- A \texttt{QueryBitmap} function is added in the \texttt{PXCMImage} function for rendering the image on the screen.

See \texttt{framework/CSharp/gesture\_viewer\_simple.cs} for an example.