INTEL® PERCEPTUAL COMPUTING SDK

Reference Manual

Face Analysis Modules

API Version 1.0

LEGAL DISCLAIMER

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT © 2011-2013, INTEL CORPORATION. ALL RIGHTS RESERVED.

Optimization Notice

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

Notice revision #20110804

Table of Contents

Intel® Perceptual Computing SDK 1

Document Conventions 1

Programming Guide 2

General Procedure 2

Face Location Data 5

Face Landmark Data 5

Face Attribute Data 6

Face Recognition 7

Interface and Function Reference 9

PXCFaceAnalysis 9

ProcessImageAsync 9

QueryFace 10

QueryProfile 11

SetProfile 11

PXCFaceAnalysis::Attribute 12

QueryData 12

QueryProfile 13

SetProfile 14

PXCFaceAnalysis::Detection 15

QueryData 15

QueryProfile 16

SetProfile 16

PXCFaceAnalysis::Landmark 17
QueryLandmarkData 18
QueryPoseData 18
QueryProfile 19
SetProfile 20
PXCFaceAnalysis::Recognition 20
CreateModel 21
DeserializeModel 22
QueryProfile 22
SetProfile 23
PXCFaceAnalysis::Recognition::Model 23
Compare 24
Serialize 25
Structure Reference 26
PXCFaceAnalysis::Attribute::ProfileInfo 26
PXCFaceAnalysis::Detection::Data 26
PXCFaceAnalysis::Detection::ProfileInfo 27
PXCFaceAnalysis::LandMark::LandmarkData 28
PXCFaceAnalysis::LandMark::PoseData 28
PXCFaceAnalysis::Landmark::ProfileInfo 29
PXCFaceAnalysis::ProfileInfo 30
PXCFaceAnalysis::Recognition::ProfileInfo 30
Enumerator Reference 32
PXCFaceAnalysis::Attribute::Index 32

PXCFaceAnalysis::Attribute::Label 33

PXCFaceAnalysis::Detection::ViewAngle 33

PXCFaceAnalysis::Landmark::Label 34

Intel® Perceptual Computing SDK

The Intel® Perceptual Computing SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The library's purpose is to lower 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 face analysis modules of the Intel® Perceptual Computing SDK Application Programming Interface (API). The other Perceptual Computing SDK 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`**.

Programming Guide

The SDK face tracking module provides a suite of face tracking algorithms including face detection, landmark detection, face recognition, and face attribute detection.

As shown in Figure 1, face detection locates the rectangular position of a face or multiple faces from an image or a video sequence. It is useful to know how many faces are in the picture and their general locations.

Landmark detection further identifies the feature points (eyes, mouth, etc.) for a given face rectangle. The eye location is of a particular interest for applications that change display perspectives based on where on the screen users are looking. Other feature points can be useful to create a face avatar or figure out the orientation of the head.

Face attribute detection analyzes the face(s) and returns the face age group, gender, and expression details. Finally, face recognition associates the given face to a name if the person is previously registered in a name database.

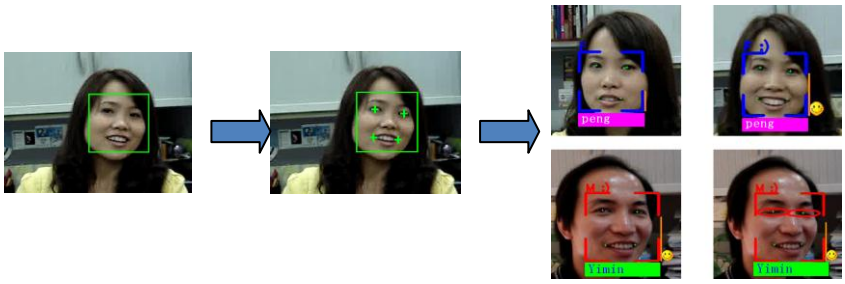

Figure 1: Face Tracking Operations

General Procedure

The application uses the following procedure to create an instance of the face tracking module and deliver data to it:

- **Locate the Module Implementation:** The application uses the **`PXCSession::CreateImpl`** function to create an instance of the module interface **`PXCFaceAnalysis`**, as illustrated in Example 1.

```
PXCFaceAnalysis *face=0;
session→CreateImpl(PXCFaceAnalysis::CUID,(void**)&face);
```

Example 1: Create a Face Tracking Module Instance

- **Initialize the Module**: The two functions for module initialization are **QueryProfile** and **SetProfile**. The former function returns available configurations. The latter sets one as the current active configuration. In Example 2, the application queries the first supported configuration; uses it to locate an input device that can provide data; and then initializes the module with the configuration. Note that the module input needs are specified as part of the **ProfileInfo** structure, which may be a depth image, and/or a color image. The utility class **UtilCapture** will locate an input device that matches the module data needs.

```
PXCFaceAnalysis::ProfileInfo pinfo;
face→QueryProfile(0,&pinfo);
UtilCapture capture(&session);
capture.LocateStreams(&pinfo.inputs);
face→SetProfile(&pinfo);
```

Example 2: Initialize a Face Tracking Module

- **Initialize Face Tracking Features**: The application needs to initialize each face tracking feature such as face detection and landmark detection, in addition to the module initialization. Each feature has its own configuration parameters. The application calls the **QueryProfile** and **SetProfile** functions on the feature interfaces, for example, **PXCFaceAnalysis::Detection**, **PXCFaceAnalysis::Landmark**, as illustrated in Example 3:

```
PXCFaceAnalysis::Detection *det=face→DynamicCast
                  <PXCFaceAnalysis::Detection>();
PXCFaceAnalysis::Detection::ProfileInfo dinfo;
det→QueryProfile(0,&dinfo);
det→SetProfile(0,&dinfo);
```

Example 3: Initialize Face Detection of the Face Tracking Module

- **Data Processing Loop**: In the loop, the application passes samples from the input device to the face tracking module. The application calls the **ProcessImageAsync** function to deliver the samples, as illustrated in Example 4. Note that the samples from the input device may contain multiple images such as a color image and a depth image. The application can use **PXCSmartArray<PXCImage>** set of functions to simplify programming.

If the application needs to switch context, the application calls the **ProcessImageAsync** function with a **NULL** pointer to reset any tracking states saved in the module.

```
PXCSmartArray<PXCImage> images;
PXCSmartSPArray sps(2);
for (;;) {
      // Get samples from input device and pass to the module
      capture.ReadStreamAsync(images.ReleaseRefs(),sps.ReleaseRef(0));
      face→ProcessImageAsync(images,sps.ReleaseRef(1));
      sps.SynchronizeEx();
      // Tracking or recognition results are ready. Now process them
      …
}
```

Example 4: Face Tracking Module Data Processing Loop

For simple face location and landmark detection usage from a camera input or a recorded file, the application can use the pipeline utility class **UtilPipeline**, a hard-coded version of the above procedure, as illustrated in Example 5. See the utility class **UtilPipeline** description for details.

```
class MyPipeline: public UtilPipeline {
public:
      MyPipeline() {
            EnableFaceLandmark();
      }
      virtual bool OnNewFrame() {
            PXCFaceAnalysis *face=QueryFace();
            … // Face tracking operations
      }
};
int wmain(int argc, wchar_t* argv[]) {
      MyPipeline mp;
      mp.LoopFrames();
      return 0;
}
```

Example 5: Face Detection Using UtilPipeline

Face Location Data

The application can use the **QueryData** function of the **PXCFaceAnalysis::Detection** interface to retrieve the face location data. There may be multiple faces in any image. The application can use the **QueryFace** function to retrieve the face identifiers of available faces in the image.

```
PXCFaceAnalysis::Detection *detector=face→DynamicCast
                        <PXCFaceAnalysis::Detection>();
for (int i=0;;i++) {
      pxcUID fid; pxcU64 ts;
      if (face→QueryFace(i,&fid,&ts)<PXC_STATUS_NO_ERROR) break;
      PXCFaceAnalysis::Detection::Data data;
      detector→QueryData(fid,&data);
      // Process data
}
```

Example 6: Retrieve Face Location Data for All Faces

Face Landmark Data

Similar to the face location data, the application can use the **QueryLandmarkData** **and** **QueryPoseData** functions of the **PXCFaceAnalysis::Landmark** interface to retrieve landmark related data, as illustrated in Example 7. The **LandmarkData** structure provides the position information for a particular landmark (or a facial feature point.) The **PoseData** structure provides the head orientation.

```
PXCFaceAnalysis::Landmark *landmark=face→DynamicCast
                        <PXCFaceAnalysis::Landmark>();
for (int i=0;;i++) {
      pxcUID fid; pxcU64 ts;
      if (face→QueryFace(i,&fid,&ts)<PXC_STATUS_NO_ERROR) break;
      PXCFaceAnalysis::Landmark::LandmarkData ldata;
      landmark→QueryLandmarkData(fid,
                  PXCFaceAnalysis::Landmark::LABEL_NOSE_TIP,
                  0, &data);
      PXCFaceAnalysis::Landmark::PoseData pdata;
      landmark→QueryPoseData(fid,&pdata);
      // Process data
}
```

Example 7: Retrieve Face Landmark Data for All Faces

Face Attribute Data

The application can use the **QueryData** functions of the **PXCFaceAnalysis::Attribute** interface to retrieve facial attribute data. The function returns the likelihood score for each attribute data. For example, in Example 8, the application retrieves the smiling likelihood score. The score is pre-filtered by the smiling threshold thus any score bigger than zero indicates a smile.

Certain attributes, such as age group or gender, have more than one attribute likelihood score. The application can use the **Index** enumerator to access each attribute in the likelihood score array, for example, **scores[INDEX_GENDER_FEMALE]** is the likelihood score of the face being a female face.

```
PXCFaceAnalysis::Attribute *attr=face→DynamicCast
                        <PXCFaceAnalysis::Attribute>();
for (int i=0;;i++) {
      pxcUID fid;
      if (face→QueryFace(i,&fid)<PXC_STATUS_NO_ERROR) break;
      pxcU32 smile;
      attr→QueryData(PXCFaceAnalysis::Attribute::LABEL_EMOTION,fid,&smile);
      if (smile>0) {
            // face fid smiled
      } else {
            // face fid not yet smile
      }
}
```

Example 8: Retrieve Facial Attribute Data for All Faces

Face Recognition

Face recognition works by comparing a picture (that contains a face of the person in front of the camera) to a set of reference pictures. The most similar reference picture wins, and the person that is associated with the reference picture is the recognized person. The application needs to index and maintain the reference picture list.

Example 9 shows a simple example of calculating the similarity of two still images. The application creates two **Model** instances of the two input images by running the **ProcessImageAsync** function, and then the **CreateModel** function. Here the **Model** instance is an abstraction of the data set that uniquely represents a face in the picture, for comparison purposes. Then the application uses the **Compare** function to calculate the similarity between the two models.

In more complicated usage scenario, the application may want to process reference pictures offline and save the reference data to some database. At run time, the application retrieves the reference data from the database and recreates the reference list. The two functions **Serialize** and **DeserializeModel** are designed for such usage.

```
PXCFaceAnalysis::Recognition::Model *createModelFromImage(PXCImage *image) {
      PXCSmartSP sp;
      face→ProcessImageAsync(&image,&sp);
      sp→Synchronize();
      pxcUID fid;
      face→QueryFace(0,&fid);
      PXCFaceAnalysis::Recognition::Model *m1=0;
      face→DynamicCast<PXCFaceAnalysis::Recognition>()→CreateModel(fid,&m1);
      return m1;
}
bool IsSimilar(PXCImage *image1, PXCImage *image2) {
      PXCFaceAnalysis::Recognition::Model *ref=createModelFromImage(image1);
      PXCFaceAnalysis::Recognition::Model *cur=createModelFromImage(image2);
      pxcU32 index;
      return cur→Compare(&ref,1,0,&index)>=PXC_STATUS_NO_ERROR;
}
```

Example 9: Calculate Similarity of Two Faces

Interface and Function Reference

This section describes SDK functions and their operations.

In each function 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.

PXCFaceAnalysis

The **PXCFaceAnalysis** interface abstracts the SDK face analysis algorithms that consist of face location detection, landmark detection, recognition, and attribute detection.

The application can use the **DynamicCast** function to derive the **Detection**, **Landmark**, **Recognition**, and **Attribute** interfaces, by using the corresponding interface identifiers.

The application can create an instance of this **PXCFaceAnalysis** interface by using the **CreateImpl** function with the interface identifier **PXCFaceAnalysis::CUID**.

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

Member Functions	Description
QueryProfile	Return the supported configurations.
SetProfile	Set the working configuration parameters.
QueryFace	Enumerate all detected faces.
ProcessImageAsync	Run face analysis on input streams.

ProcessImageAsync

Syntax

```
pxcStatus ProcessImageAsync(PXCImage **images, PXCScheduler::SyncPoint
**sp);
```

Parameters

`images`	A still image or a frame of the input video sequence (max. of three channels supported).

`sp`	The SP (synchronization point), to be returned.

Description

This function performs a suite of face analysis algorithms, including face location detection, landmark detection, recognition, and attribute detection on the input streams.

This function is asynchronous. The analysis results are available only after the application synchronizes the SPs.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.

Change History

This function was introduced in SDK API 1.0.

QueryFace

Syntax

```
pxcStatus QueryFace(pxcU32 fidx, pxcUID *fid, pxcU64 *timeStamp);
```

Parameters

`fidx`	Zero-based index to enumerate all detected faces.
`fid`	The face identifier, to be returned.
`timeStamp`	The detection time stamp, in 100 ns.

Description

This function enumerates all detected faces.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	Failed to locate the detected face by the index.

Change History

This function was introduced in SDK API 1.0.

QueryProfile

Syntax

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

Parameters

pidx	Zero-based profile index; use **WORKING_PROFILE**, the default value, to access the current working configuration parameters.
pinfo	The configuration parameters in the **ProfileInfo** structure, to be returned.

Description

This function enumerates the supported configuration parameters of the face analysis algorithm. Each profile consists of a set of supported configuration parameters. The application can increase the **pidx** value to loop through multiple configurations that the algorithm may support.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE	Failed to find the specified index of the configuration set.

Change History

This function was introduced in SDK API 1.0.

SetProfile

Syntax

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

Parameters

pinfo	The configuration parameters in the **ProfileInfo** structure.

Description

This function configures the face analysis algorithm with the specified parameters. The profile parameters do not have to exactly match what the **QueryProfile** function returns.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_PARAMS_UNSUPPORTED	There are unsupported parameters in the configuration.

Change History

This function was introduced in SDK API 1.0.

PXCFaceAnalysis::Attribute

The **Attribute** interface abstracts the face attribute detection algorithm and is part of the SDK face analysis algorithms.

The application can derive an instance of the **Attribute** interface from the **PXCFaceAnalysis** interface by using the **DynamicCast** function. The corresponding interface identifier is **PXCFaceAnalysis::Attribute::CUID**.

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

Member Functions	Description
QueryProfile	Return the supported configuration parameters.
SetProfile	Set the working configuration parameters.
QueryData	Return the detected face attributes.

QueryData

Syntax

```
pxcStatus QueryData(Label label, pxcUID fid, pxcU32 *scores);
```

Parameters

Label	The attribute label; see the **Label** enumerator for

	definitions.
`fid`	The face identifier.
`scores`	The face attribute scores, to be returned.

Description

This function returns the detected face attribute scores for a given face identifier. The application must pre-allocate the **`scores`** array, which follows the defined order as in the **`Index`** enumerator.

The returned likelihood scores are filtered by the attribute threshold. For example, for smiling, any likelihood score bigger than zero indicates a smile.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The face identifier was not found.

Change History

This function was introduced in SDK API 1.0.

QueryProfile

Syntax

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

Parameters

`label`	The attribute label; see the **`Label`** enumerator for definitions.
`pidx`	Zero-based profile index; use **`WORKING_PROFILE`**, the default value, to retrieve the current working configuration parameters.
`pinfo`	The configuration parameters in the **`ProfileInfo`** structure, to be returned.

Description

This function enumerates the supported configuration parameters. The application increases **`pidx`** to loop through multiple configurations that the algorithm may

support.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE	Failed to find the specified index of the configuration set.

Change History

This function was introduced in SDK API 1.0.

SetProfile

Syntax

```
pxcStatus SetProfile(Label label, ProfileInfo *pinfo);
```

Parameters

label	The attribute label; see the **Label** enumerator for definitions.
pinfo	The configuration parameters in the **ProfileInfo** structure.

Description

This **SetProfile** function sets the configuration parameters for the face attribute detection algorithm. The specified profile parameters do not have to exactly match what is returned from the **QueryProfile** function.

Note that the configuration parameters do not take effect until the application synchronizes after the **ProcessImageAsync** function.

The application must set the working parameters for each face attribute at least once or the face attribute detection step is disabled.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_PARAMS_UNSUPPORTED	There are unsupported parameters in the configuration.

Change History

This function was introduced in SDK API 1.0.

PXCFaceAnalysis::Detection

The **Detection** interface abstracts the face location detection algorithm for a still image or a video sequence, and is part of the SDK face analysis algorithms. In the video sequence case, the algorithm also tracks the detected faces in subsequent video frames.

The application can derive an instance of the **Detection** interface from the **PXCFaceAnalysis** interface by using the **DynamicCast** function. The corresponding interface identifier is **PXCFaceAnalysis::Detection::CUID**.

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

Member Functions	Description
QueryProfile	Query the supported face detection configurations.
SetProfile	Set the face detection configuration parameters.
QueryData	Retrieve the location parameters of a detected face.

QueryData

Syntax

```
pxcStatus QueryData(pxcUID fid, Data *data);
```

Parameters

`fid`	The face identifier.
`data`	The face location parameters in the **Data** structure, to be returned.

Description

This function returns the face location parameters of a detected face.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The specified face identifier was not found.

Change History

This function was introduced in SDK API 1.0.

QueryProfile

Syntax

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

Parameters

pidx	Zero based profile index. Use **WORKING_PROFILE**, the default value, to access the current working configuration parameters.
pinfo	Configuration parameters in the **ProfileInfo** structure, to be returned.

Description

This **QueryProfile** function enumerates supported configuration parameters. Each set of configurations describes a different supported capability. The application needs to call the **SetProfile** function to set the desired configuration parameters as part of the algorithm initialization process.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE	No profile parameters found with the given index.

Change History

This function was introduced in SDK API 1.0.

SetProfile

Syntax

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

Parameters

pinfo	Configuration parameters in the **ProfileInfo** structure.

Description

This **SetProfile** function sets the configuration parameters for the face detection algorithm. The specified profile parameters do not have to exactly match what is returned from the **QueryProfile** function.

The application must set the working parameters for face detection at least once (for the first frame) or the face detection step is disabled. If the application configures landmark detection, recognition, or attribute detection that implicitly enables face detection/tracking, the detection/tracking algorithm will work on default parameters.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_PARAMS_UNSUPPORTED	There are unsupported parameters in the specified configuration.

Change History

This function was introduced in SDK API 1.0.

PXCFaceAnalysis::Landmark

The **Landmark** interface abstracts the face landmark detection algorithm and is part of the SDK face analysis algorithms.

The application can derive an instance of the **Landmark** interface from the **PXCFaceAnalysis** interface by using the **DynamicCast** function. The corresponding interface identifier is **PXCFaceAnalysis::Landmark::CUID**.

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

Member Functions	Description
QueryProfile	Return the supported configuration parameters.
SetProfile	Set the working configuration parameters.
QueryLandmarkData	Return the landmark detection parameters.
QueryPoseData	Return the face pose parameters.

QueryLandmarkData

Syntax

```
pxcStatus QueryLandmarkData(pxcUID fid, Label label, pxcU32 lidx,
LandmarkData *data);
```

```
pxcStatus QueryLandmarkData(pxcUID fid, Label label, LandmarkData
*data_array);
```

Parameters

`fid`	The face identifier.
`label`	The landmark name. See the **Label** enumerator for definitions.
`lidx`	Zero-based index to enumerate all landmarks if the **label** parameter is a landmark group, such as **LABEL_6PTS** or **LABEL_7PTS**. For individual landmarks, this parameter must be zero.
`data`	The individual landmark data in the **LandmarkData** structure, to be returned.
`data_array`	The array of landmark data, to be returned. The **label** parameter must be a landmark group.

Description

This function returns detected individual or an array of landmark parameters.

Return Status

`PXC_STATUS_NO_ERROR`	The function completed successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The requested landmarks were not available.

Change History

This function was introduced in SDK API 1.0.

QueryPoseData

Syntax

```
pxcStatus QueryPoseData(pxcUID fid, PostData *pose);
```

Parameters

`fid`	The face identifier.
`pose`	The face pose parameters in the **PoseData** structure, to be returned.

Description

This function returns the face pose parameters.

Return Status

`PXC_STATUS_NO_ERROR`	The function completed successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The requested face identifier was not available.
`PXC_STATUS_FEATURE_UNSUPPORTED`	The SDK is unable to determine the face position.

Change History

This function was introduced in SDK API 1.0.

QueryProfile

Syntax

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

Parameters

`pidx`	Zero-based profile index. Use **WORKING_PROFILE**, the default value, to access the current working parameters.
`pinfo`	Configuration parameters in the **ProfileInfo** structure.

Description

This function enumerates supported configuration parameters by index. The application can increase the **pidx** value to enumerate multiple supported configuration parameters.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_ITEM_UNAVAILABLE	No more configuration parameters.

Change History

This function was introduced in SDK API 1.0.

SetProfile

Syntax

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

Parameters

pinfo	Configuration parameters in the **ProfileInfo** structure.

Description

This **SetProfile** function sets the configuration parameters for the face landmark detection algorithm. The specified profile parameters do not have to exactly match what is returned from the **QueryProfile** function.

The application must set the working parameters for face landmark detection at least once or the face landmark detection step is disabled. If the application configures face recognition or attribute detection that implicitly enables face landmark detection, the landmark algorithm will work on default parameters.

Return Status

PXC_STATUS_NO_ERROR	The function completed successfully.
PXC_STATUS_PARAMS_UNSUPPORTED	There are unsupported parameters in the configuraion.

Change History

This function was introduced in SDK API 1.0.

PXCFaceAnalysis::Recognition

The **Recognition** interface abstracts the face recognition algorithm and is part of the SDK face analysis algorithms.

The application can derive an instance of the **Recognition** interface from the **PXCFaceAnalysis** interface by using the **DynamicCast** function. The corresponding interface identifier is **PXCFaceAnalysis::Recognition::CUID**.

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

Member Functions	Description
QueryProfile	Return the supported configuration parameters.
SetProfile	Set the working configuration parameters.
CreateModel	Create a **Model** instance for the current picture.
DeserializeModel	Create a **Model** instance of the reference model for subsequent face recognition operations.

CreateModel

Syntax

```
pxcStatus CreateModel(pxcUID fid, Model **model);
```

Parameters

`fid`	The face identifier.
`model`	The **Model** instance, to be returned.

Description

This function returns a **Model** instance that represents the specified face in the picture. The application must pre-process the picture with the **PXCFaceAnalysis** interface.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	The specified face identifier is not available.

Change History

This function was introduced in SDK API 1.0.

DeserializeModel

Syntax

```
pxcStatus DeserializeModel(pxcBYTE *buffer, Model **model);
```

Parameters

`buffer`	The buffer that contains the serialized model representation.
`model`	The **Model** instance, to be returned.

Description

This function recreates the **Model** instance based on a previously serialized model representation.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.

Change History

This function was introduced in SDK API 1.0.

QueryProfile

Syntax

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

Parameters

`pidx`	The zero-based profile index; use **WORKING_PROFILE**, **the default value**, to access the current working parameters.
`pinfo`	The profile information in the **ProfileInfo** structure, to be returned.

Description

This function enumerates the configuration parameters for the face recognition algorithm. The application needs to increase the profile index to enumerate all supported configurations.

Return Status

PXC_STATUS_NO_ERROR	The function returned successfully.
PXC_STATUS_ITEM_UNAVAILABLE	The requested configuration is not available.

Change History

This function was introduced in SDK API 1.0.

SetProfile

Syntax

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

Parameters

pinfo	The specified profile parameters, in the **ProfileInfo** structure.

Description

This **SetProfile** function sets the configuration parameters for the face recognition algorithm. The specified profile parameters do not have to exactly match what is returned from the **QueryProfile** function.

The application must set the working parameters for face recognition at least once or the face recognition step is disabled.

Return Status

PXC_STATUS_NO_ERROR	The function returned successfully.
PXC_STATUS_PARAM_UNSUPPORTED	Unsupported parameters in the configuration.

Change History

This function was introduced in SDK API 1.0.

PXCFaceAnalysis::Recognition::Model

The **Model** interface abstracts the mathematical model that is used for face recognition.

The application can use the **CreateModel** function or the **DeserializeModel** function to create an instance of the **Model** interface.

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

Member Functions	Description
Serialize	Serialize the model representation for storing to a file.
Compare	Compare the model to a list of reference models for similarity.

Compare

Syntax

```
pxcStatus Compare(Model **models, pxcU32 nmodels, pxcF32 *scores, pxcU32
*index);
```

Parameters

`models`	The pointer to a list of the **Model** instances.
`nmodels`	The number of models.
`scores`	The pointer to an array of likelihood scores, to be returned. The score indicates the similarity of the model to the corresponding model in the list.
`index`	The index of the most similar model, to be returned.

Description

This function compares the model instance with the specified list of models and returns the similarity scores.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.
`PXC_STATUS_ITEM_UNAVAILABLE`	There is no model similar to the current model. The index is not returned.

Change History

This function was introduced in SDK API 1.0.

Serialize

Syntax

```
pxcStatus Serialize(pxcBYTE *buffer);
```

Parameters

`Buffer`	The buffer, to be filled, that is sufficient to store a serialized **`Model`** representation.

Description

This function serializes the current model internal representation and copies to an external buffer. The buffer size is returned as part of the **`ProfileInfo`** structure.

Return Status

`PXC_STATUS_NO_ERROR`	The function returned successfully.

Change History

This function was introduced in SDK API 1.0.

Structure Reference

In the following structure references, all reserved fields must be zero.

PXCFaceAnalysis::Attribute::ProfileInfo

Definition

```
struct ProfileInfo {
      pxcU32      threshold;
      pxcBool     smoothing;
};
```

Description

The **`ProfileInfo`** structure describes the configuration parameters of the facial attribute detection algorithm.

Members

`threshold`	The detection threshold from 0 to 100. The application needs to compare the detection result with this threshold to determine whether a profile is positive.
`smoothing`	A boolean value if the algorithm should perform smoothing during attribute detection.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::Detection::Data

Definition

```
struct Data {
      PXCRectU32  rectangle;
      pxcUID      fid;
      pxcU32      confidence;
```

```
        ViewAngle    viewAngle;
        pxcU64       reserved[4];
};
```

Description

The **Data** structure describes the detected face parameters.

Members

`fid`	The face identifier.
`rectangle`	The detected face rectangle coordinates.
`confidence`	The confidence score from 0 to 100.
`viewAngle`	The face view angle. See the **ViewAngle** enumerator for definitions.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::Detection::ProfileInfo

Definition

```
struct ProfileInfo {
        PXCSizeU32              minFaceSize;
        PXCSizeU32              maxFaceSize;
        pxcU64                  reserved;
        ViewAngle               viewAngles;
};
```

Description

The `ProfileInfo` structure describes configuration parameters for the face detection algorithm.

Members

`viewAngles`	The supported view angle; see definitions in the **ViewAngle** enumerator.
`minFaceSize`	The minimum face size that the algorithm can recognize.
`maxFaceSize`	The maximum face size that the algorithm can recognize.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::LandMark::LandmarkData

Definition

```
struct LandmarkData {
      PXCPoint3DF32     position;
      pxcUID            fid;
      Label             label;
      pxcU32            lidx;
};
```

Description

The **LandmarkData** structure describes the detected landmark points.

Members

fid	The face identifier.
label	The landmark label. See the **Label** enumerator for details.
lidx	Reserved.
position	The three dimensional location parameters of the landmark.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::LandMark::PoseData

Definition

```
struct PoseData {
      pxcUID            fid;
      pxcF32            yaw, roll, pitch;
```

```
};
```

Description

The **PoseData** structure describes face pose parameters.

Members

fid	The face identifier.
yaw	The yaw value.
roll	The roll value.
pitch	The pitch value.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::Landmark::ProfileInfo

Definition

```
struct ProfileInfo {
      Label        labels;
};
```

Description

The **ProfileInfo** structure describes the configuration parameters of the SDK landmark detection algorithm.

Members

labels	All supported landmark points. See definitions in the **Label** enumerator.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::ProfileInfo

Definition

```
struct ProfileInfo {
      PXCCapture::VideoStream::DataDesc   inputs;
      pxcBool            ifTracking;
};
```

Description

The **`ProfileInfo`** structure describes the configuration parameters of the facial analysis algorithm.

Members

`inputs`	The input data requirements, described in the **`PXCCapture::VideoStream::DataDesc`** structure.
`ifTracking`	A boolean value if the analysis is for a still image or for a video sequence.

Change History

This structure is introduced in SDK API 1.0.

PXCFaceAnalysis::Recognition::ProfileInfo

Definition

```
struct ProfileInfo {
      pxcU32             modelSize;
      pxcF32             threshold;
      pxcU32             reserved[6];
};
```

Description

The **`ProfileInfo`** structure describes the SDK face recognition algorithm configuration parameters.

Members

`modelSize`	The model size in bytes after serialization.
`threshold`	The threshold for the similarity score. If a similarity score is bigger than the threshold, the two comparison models are likely from the same person.

Change History

This structure is introduced in SDK API 1.0.

Enumerator Reference

PXCFaceAnalysis::Attribute::Index

Description

The **Index** enumerator defines the array index for each attribute group. The application can use this enumerator as an array index to access the attribute scores.

Name/Description

Array index for the age group scores

INDEX_AGE_GROUP_BABY	Baby 0-2 years old.
INDEX_AGE_GROUP_TODDLER	Toddler 2-3 years old.
INDEX_AGE_GROUP_YOUTH	Youth 3-15 years old.
INDEX_AGE_GROUP_ADULT	Adult 16-60 years old.
INDEX_AGE_GROUP_SENIOR	Senior 60+ years old.

Array index for the gender scores

INDEX_GENDER_MALE	Male
INDEX_GENDER_FEMALE	Female

Array index for the eye-closed scores

INDEX_EYE_CLOSED_LEFT	Left eye is closed.
INDEX_EYE_CLOSED_FEMALE	Right eye is closed.

Array index for the emotion scores

INDEX_EMOTION_SMILE	Smile

Change History

This enumerator is introduced in SDK API 1.0.

PXCFaceAnalysis::Attribute::Label

Description

The **`Label`** enumerator itemizes different facial attribute names.

Name/Description

`LABEL_AGE_GROUP`	The algorithm detects age group. Use the **`Agegroup`** index to access the likelihood scores.
`LABEL_GENDER`	The algorithm detects gender.
`LABEL_EMOTION`	The algorithm detects emotion. Use the **`Emotion`** index to access the likelihood scores.
`LABEL_EYE_CLOSED`	The algorithm detects eye closed.

Change History

This enumerator is introduced in SDK API 1.0.

PXCFaceAnalysis::Detection::ViewAngle

Description

The **`ViewAngle`** enumerator enumerates face view angles. In configuration query, the SDK returns a bit-OR'ed value of all supported view angles.

Name/Description

`VIEW_ANGLE_0`	The face is looking away from the camera to the left about 90 degrees.
`VIEW_ANGLE_45`	The face is looking away from the camera to the left about 45 degrees.
`VIEW_ANGLE_FRONTAL`	The face is facing the camera.
`VIEW_ANGLE_135`	The face is looking away from the camera to the right about 45 degrees.
`VIEW_ANGLE_180`	The face is looking away from the camera to the right about 90 degrees.
`VIEW_ANGLE_HALF_MULTI`	Equivalent to the bit-OR'ed value of view angle 45 degrees or frontal, or 135 degrees.

VIEW_ANGLE_MULTI	Equivalent to the bit-OR'ed value of view angle 0 degrees, 45 degrees, frontal, 135 degrees, and 180 degrees.
VIEW_ANGLE_OMNI	The SDK supports all above tiling and yawing positions in face detection.

Change History

This enumerator is introduced in SDK API 1.0.

PXCFaceAnalysis::Landmark::Label

Description

The **Label** enumerator uses bit-OR'ed values to list all landmark points that the SDK face landmark detection algorithm supports.

Name/Description

Individual Landmarks

LABEL_LEFT_EYE_OUTER_CORNER	The left eye outer corner feature point.
LABEL_LEFT_EYE_INNER_CORNER	The left eye inner corner feature point.
LABEL_RIGHT_EYE_OUTER_CORNER	The right eye outer corner feature point.
LABEL_RIGHT_EYE_INNER_CORNER	The right eye inner corner feature point.
LABEL_MOUTH_LEFT_CORNER	The mouth left corner feature point.
LABEL_MOUTH_RIGHT_CORNER	The mouth right corner feature point.
LABEL_NOSE_TIP	The nose tip feature point.

Landmark Group

LABEL_6POINTS	6-point landmark group including all of the above except the nose tip.
LABEL_7POINTS	7-point landmark group including all of the above individual landmarks.
LABEL_SIZE_MASK	Use this mask **AND** the landmark group to obtain the number of landmarks in the group.

Change History

This enumerator is introduced in SDK API 1.0.