



INTEL® PERCEPTUAL COMPUTING SDK

How to Use the Face Analysis Module



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](http://www.intel.com).

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

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

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

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

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



Optimization Notice

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

Notice revision #20110804



Table of Contents

Intel® Perceptual Computing SDK.....	1
General Description	1
Procedure for a Simple Face Analysis Application	3
Project Setup and Configuration	4
Code Explanation	4
Procedure for Face Analysis	7
1 st Step	7
2 nd Step	7
3 rd Step.....	8
4 th Step.....	9
5 th Step.....	10
Exercise: Putting it all together	11

Intel® Perceptual Computing SDK

The Intel® Perceptual Computing SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The SDK provides a suite of face analysis algorithms including face location detection, landmark detection, face recognition and face attribute detection.

This tutorial shows how to use the SDK for face location detection and landmark detection, and how to write an application for these face analysis modules.

General Description

As shown in Figure 1, the face detection algorithm locates the rectangle position of a face or multiple faces from an image or a video sequence in real-time capture or playback mode.



Figure 1: Face location detection, the number 100 indicates the face ID.

In Figure 2, the landmark detection algorithm locates the 6-point or 7-point landmarks namely, the outer and inner corners of the eyes, the tip of the nose, and the outer corners of the mouth.



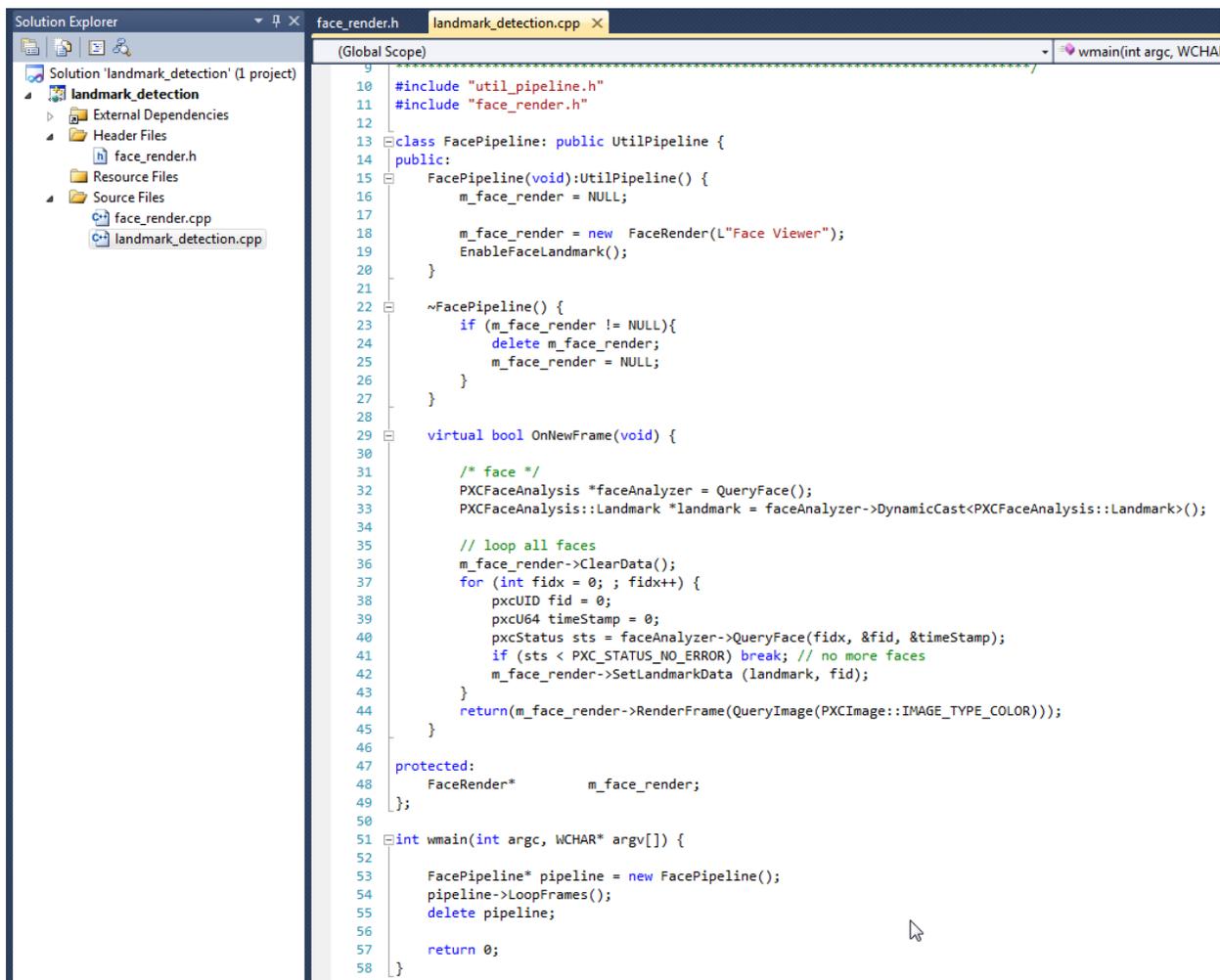
Figure 2: Face landmark detection, 7 point landmarks.

Procedure for a Simple Face Analysis Application

It is easy to create a simple face analysis application that will read color video from the camera, detect landmarks, track the faces in each frame, and render them.

The code shown in Figure 3 uses the `UtilPipeline` utility class for a simplified asynchronous pipeline implementation and the `FaceRender` class for rendering. The application will first initialize the pipeline and enable landmark detection. Then it will process each frame using the `UtilPipeline::LoopFrames` function. To observe the results, the application will render each frame with the detection and landmark data using the `FaceRender` utility.

When the application is running, you should see the color rendering window with the landmark points detected as seen in Figure 2. When you click on the close button on the top, the application closes.



```

9
10 #include "util_pipeline.h"
11 #include "face_render.h"
12
13 class FacePipeline: public UtilPipeline {
14 public:
15     FacePipeline():UtilPipeline() {
16         m_face_render = NULL;
17
18         m_face_render = new FaceRender(L"Face Viewer");
19         EnableFaceLandmark();
20     }
21
22     ~FacePipeline() {
23         if (m_face_render != NULL){
24             delete m_face_render;
25             m_face_render = NULL;
26         }
27     }
28
29     virtual bool OnNewFrame(void) {
30
31         /* face */
32         PXCFaceAnalysis *faceAnalyzer = QueryFace();
33         PXCFaceAnalysis::Landmark *landmark = faceAnalyzer->DynamicCast<PXCFaceAnalysis::Landmark>();
34
35         // loop all faces
36         m_face_render->ClearData();
37         for (int fidx = 0; ; fidx++) {
38             pxcUID fid = 0;
39             pxcU64 timeStamp = 0;
40             pxcStatus sts = faceAnalyzer->QueryFace(fidx, &fid, &timeStamp);
41             if (sts < PXC_STATUS_NO_ERROR) break; // no more faces
42             m_face_render->SetLandmarkData (landmark, fid);
43         }
44         return(m_face_render->RenderFrame(QueryImage(PXCImage::IMAGE_TYPE_COLOR)));
45     }
46
47 protected:
48     FaceRender*          m_face_render;
49 };
50
51 int wmain(int argc, WCHAR* argv[]) {
52
53     FacePipeline* pipeline = new FacePipeline();
54     pipeline->LoopFrames();
55     delete pipeline;
56
57     return 0;
58 }

```

Figure 3: A simple landmark detection code.

Project Setup and Configuration

For project setup and configuration, please follow the steps in the “Getting Started” tutorial to create and configure the project. After completing the project setup and configuration, from the $\$(PCSDK_DIR)/samples/common/include$ and $\$(PCSDK_DIR)/samples/common/src$ add the “face_render.h” and “face_render.cpp” files to the project, as shown in Figure 4.

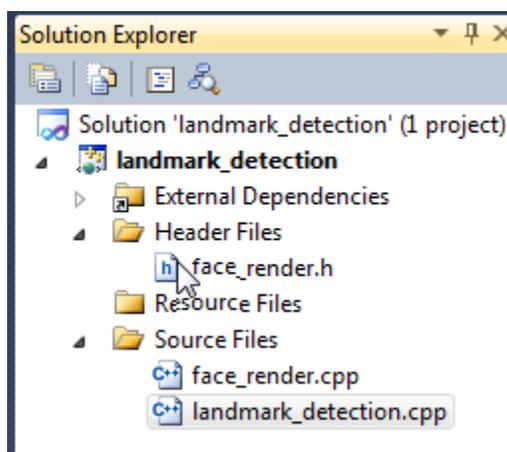


Figure 4: Files to add to the project

Code Explanation

The first step is to use the `UtilPipeline` class to build a very simple application and define a new class `FacePipeline` which inherits from this utility class, shown in Example 1. The `FacePipeline` class is needed to render the face landmarks since this functionality is not a part of the `UtilPipeline` utility class. So the constructor declares a new `FaceRender` and the destructor cleans it.



```
#include "util_pipeline.h"
#include "face_render.h"

class FacePipeline: public UtilPipeline {
public:
    FacePipeline(void):UtilPipeline() {
        m_face_render = NULL;

        // Create a face renderer
        m_face_render = new FaceRender(L"Face Viewer");

        // Enable the face landmark detector
        EnableFaceLandmark();
    }

    ~FacePipeline() {
        if (m_face_render != NULL) delete m_face_render;
    }

protected:
    FaceRender*      m_face_render;
};
```

Example 1: FacePipeline class initial framework

In order to render the landmarks as shown in Example 2, the application will need to implement the `UtilPipeline::OnNewFrame` function that renders face analysis detection data using `FaceRender` utility. The `OnNewFrame` function uses `UtilPipeline::QueryFace()` to query the face analyzer class. Then the face analyzer uses its `PXCFaceAnalysis::QueryFace()` function to query the detected face. If a face is detected, the application must query the detection data or the landmark data after dynamically casting to the respective module `PXCFaceAnalysis::Landmark`. Finally, a call to `FaceRender::SetLandmarkData` is used for setting the face analysis data that will be rendered when calling the `FaceRender::RenderFrame` routine.



```
virtual bool OnNewFrame(void) {  
  
    /* face */  
    PXCFaceAnalysis *faceAnalyzer = QueryFace();  
    PXCFaceAnalysis::Landmark *landmark =  
        faceAnalyzer->DynamicCast<PXCFaceAnalysis::Landmark>();  
  
    // loop all faces  
    m_face_render->ClearData();  
    for (int fidx = 0; ; fidx++) {  
        pxcUID fid = 0;  
        pxcU64 timeStamp = 0;  
        pxcStatus sts = faceAnalyzer->QueryFace(fidx, &fid,  
&timeStamp);  
        if (sts < PXC_STATUS_NO_ERROR) break; // no more faces  
        m_face_render->SetLandmarkData (landmark, fid);  
    }  
    return(m_face_render->RenderFrame(  
QueryImage(PXCImage::IMAGE_TYPE_COLOR) ));  
}
```

Example 2: OnNewFrame Implementation

To process each frame of the captured video sequence shown in Example 3, the main function simply calls **LoopFrames()** function to capture and process each frame. The **LoopFrames** function also calls the **OnNewFrame** function for retrieving and rendering the detection data.

```
FacePipeline* pipeline = new FacePipeline();  
pipeline->LoopFrames();  
delete pipeline;
```

Example 3: Process each frame

A more detailed example of the **PxcFaceAnalysis** module without utilizing the **UtilPipeline** class is given in the following section.



Procedure for Face Analysis

The application uses the following procedure for the face detection operations on a still image or a video sequence:

1st Step

In Example 4, the SDK session is the very first object the face detection application must create to hold all I/O or algorithm modules. This is created by calling the `PXCSession_Create` function. Then the application uses the session functions to create a module instance of the face detection algorithms, which is a part of the `PXCFaceAnalysis` interface. So, the application first creates an instance of the face analysis interface `PXCFaceAnalysis` by calling the `PXCSession::CreateImpl` function with the interface identifier `PXCFaceAnalysis::CUID`.

```
// Create a session
PXCSession *session;
PXCSession_Create(&session);

// Create Face analyzer interface
PXCFaceAnalysis *faceAnalyzer;
session->CreateImpl(PXCFaceAnalysis::CUID, (void**) &faceAnalyzer);
```

Example 4: creating the module instance

2nd Step

The second step is to configure the `PXCFaceAnalysis` interface as seen in Example 5. The application enumerates supported configurations by using the `QueryProfile` function. Each profile describes a supported configuration. The application sets the desired configuration parameters by using the `SetProfile` function.

```
// Configure the face analysis interface
PXCFaceAnalysis::ProfileInfo faInfo;
faceAnalyzer->QueryProfile(0, &faInfo);
faceAnalyzer->SetProfile(&faInfo);
```

Example 5: Module configuration

Profile '0' is the first profile and is usually the default profile of the interface. The `PXCFaceAnalysis::ProfileInfo` describes the video streams capture info and general parameters that are common to the different face analysis modules (detection, landmark, etc.). Thus, after `QueryProfile`, the capture device and video streams can be configured for face



analysis procedures according to the enumerated `PXCFaceAnalysis::ProfileInfo`. Therefore, the `PXCFaceAnalysis` interface and capture device configurations need to be done jointly as follows in Example 6:

```
// Configure the face analysis interface
PXCFaceAnalysis::ProfileInfo faInfo;
faceAnalyzer->QueryProfile(0, &faInfo);

// Find capture device
UtilCaptureFile capture(session,0,0);
capture.LocateStreams(&faInfo.inputs);

// Set the face analysis interface
faceAnalyzer->SetProfile(&faInfo);
```

Example 6: Locate the video stream

Here the `UtilCaptureFile` tool is utilized for a simplified approach of using the capture devices and setting the appropriate capturing stream.

3rd Step

At this point, the application may selectively configure a few or all of the analysis modules, such as face location detection, face landmark detection, and other implemented face analysis modules. The procedure of configuring these modules is similar. As an example, for configuring the face location detection, the application derives the detection interface from the `PXCFaceAnalysis` interface by using the `DynamicCast` function, and then calls the `QueryProfile` and `SetProfile` functions shown in Example 7.

```
// Configure the face detector interface
PXCFaceAnalysis::Detection *faceDetector;
faceDetector=faceAnalyzer->DynamicCast<PXCFaceAnalysis::Detection>();

// Set the face detector profile
PXCFaceAnalysis::Detection::ProfileInfo dInfo={0};
faceDetector->QueryProfile(0, &dInfo);
faceDetector->SetProfile(&dInfo);
```

Example 7: Detection module configuration

The `PXCFaceAnalysis::Detection::ProfileInfo` describes the supported view angle of the detection algorithm. The default `ViewAngle` is set to `VIEW_ANGLE_MULTI`, which is equivalent to the bit-OR'ed value of view angle 0 degrees, 45 degrees, frontal (facing the camera), 135 degrees, and 180 degrees. Similarly, the same steps are followed for configuring the face landmark detection module as presented in Example 8.



```
// Configure the face detector interface
PXCFaceAnalysis::Landmark *landmarkDetector;
landmarkDetector=faceAnalyzer->DynamicCast<PXCFaceAnalysis::Landmark>();

// Set the face landmark detector profile
PXCFaceAnalysis::Landmark::ProfileInfo lInfo={0};
landmarkDetector->QueryProfile(1, &lInfo);
landmarkDetector->SetProfile(&lInfo);
```

Example 8: Landmark module configuration

4th Step

The application performs face analysis by calling the **ProcessImageAsync** function. If the face analysis is on a still image, the application provides the image at the input. If the face analysis is on a video sequence, the application provides one video frame at a time in the main loop. The **ProcessImageAsync** function returns an SP, which the application can synchronize with before retrieving the analysis results. This is shown in Example 9 as follows:

```
PXCSmartArray<PXCIImage> images; // declare image interface
PXCSmartSPArray sps(2); //declare Sync points for asynchronous execution

// Read frame
pxcStatus sts = capture.ReadStreamAsync(images,&sps[0]);

// Process frame using face detection
sts = faceAnalyzer->ProcessImageAsync(images, &sps[1]);
if (sts<PXC_STATUS_NO_ERROR) break;

// Synchronize both sync points
sts = sps.SynchronizeEx();
if (sps[0]->Synchronize(0)<PXC_STATUS_NO_ERROR) break; // test for EOF
```

Example 9: Asynchronous processing

For each captured frame, an asynchronous pipeline is executed that consists of reading a frame and then processing the current frame using any of the configured face analysis algorithms. Note that nothing is processed in the asynchronous pipeline until the Sync Points are synchronized by calling the **SynchronizeEx** function. The Asynchronous pipeline for Face Landmark detection is shown in Figure 5.

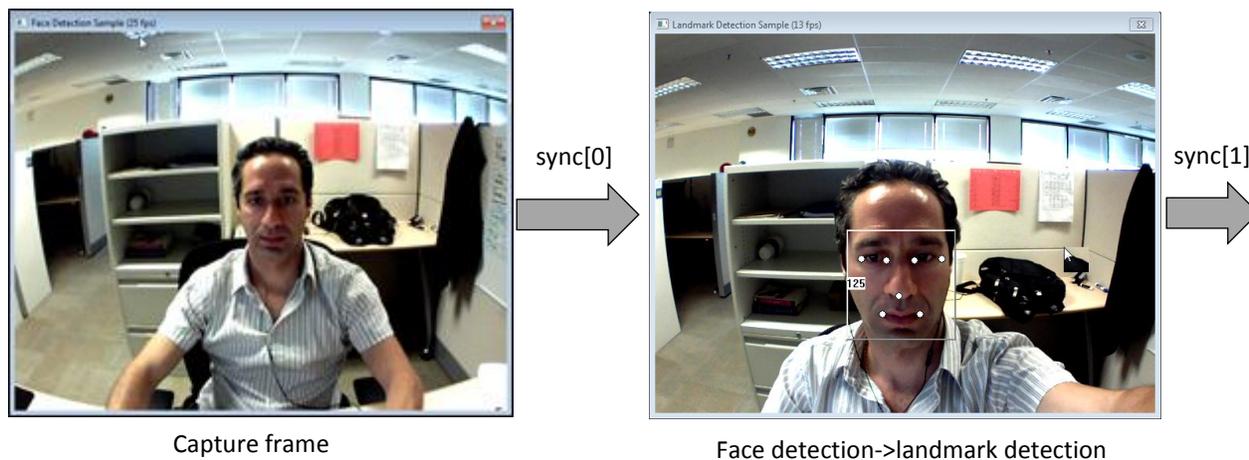


Figure 5: Face landmark detection pipeline

5th Step

To retrieve the face detection results, the application needs to use `QueryData` function in the Detection interface shown in Example 10. `QueryFace` is first called to enumerate all the detected faces during face analysis processing.

```

for (int fidx = 0; ; fidx++) {
    pxcUID fid = 0; pxcU64 timeStamp = 0;
    sts = faceAnalyzer->QueryFace(fidx, &fid, &timeStamp);
    if (sts < PXC_STATUS_NO_ERROR) break; // no more faces detected

    PXCFaceAnalysis::Detection::Data face_data;
    faceDetector->QueryData(fid, &face_data);
}

```

Example 10: Query the detection data

To retrieve the face landmark detection results, the application needs to `QueryData` for the labels in the already set profile, as follows in Example 11:

```
for (int fidx = 0; ; fidx++) {  
    pxcUID fid = 0; pxcU64 timeStamp = 0;  
    sts = faceAnalyzer->QueryFace(fidx, &fid, &timeStamp);  
    if (sts < PXC_STATUS_NO_ERROR) break; // no more faces detected  
  
    PXCFaceAnalysis::Landmark::ProfileInfo lInfo={0};  
    landmark->QueryProfile(&lInfo);  
  
    PXCFaceAnalysis::Landmark::LandmarkData data[7];  
    pxcStatus sts=landmark->QueryLandmarkData(fid,lInfo.labels,&data[0]);  
}
```

Example 11: Retrieve the face Landmark data

Exercise: Putting it all together

After explaining key parts of the application above, run the landmark detection sample code shown in Example 12. Copy the code and construct a project following the section “Project Setup and Configuration” explained previously in this tutorial. Similarly, add the following files shown in Figure 6 to the project.

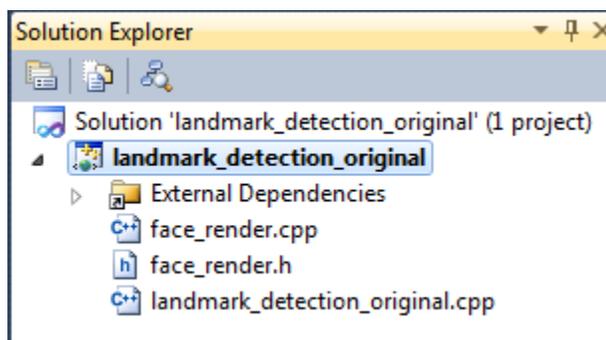


Figure 6: Project files to be added



```
#include "pxcsession.h"
#include "pxccapture.h"
#include "pxcsmartptr.h"
#include "face_render.h"
#include "util_capture_file.h"
#include "util_cmdline.h"
#include "pxcface.h"

int wmain(int argc, wchar_t* argv[])
{
    // Create a session
    PXCSmartPtr<PXCSession> session;
    pxcStatus sts=PXCSession_Create(&session);
    if (sts<PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Failed to create the SDK session\n");
        return 3;
    }

    UtilCmdLine cmdl(session);
    if (!cmdl.Parse(L"-sdname-nframes-file-record",argc,argv)) return 3;

    // Init Face analyzer
    PXCSmartPtr<PXCFaceAnalysis> faceAnalyzer;
    sts=session->CreateImpl(cmdl.m_iuid, PXCFaceAnalysis::CUID,
(void*)&faceAnalyzer);
    if (sts<PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Failed to locate a face module");
        return 3;
    }

    // Retrieve the input requirements
    PXCFaceAnalysis::ProfileInfo faInfo;
    faceAnalyzer->QueryProfile(0, &faInfo);

    // Find capture device
    UtilCaptureFile capture(session,cmdl.m_recordedFile,cmdl.m_bRecord);
    if (cmdl.m_sdname) capture.SetFilter(cmdl.m_sdname); /*L"Integrated
Camera"*/
    sts=capture.LocateStreams(&faInfo.inputs);
    if (sts<PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Failed to locate an input device that matches
input for face analysis\n");
        return 3;
    }
    faceAnalyzer->SetProfile(&faInfo);

    // Create detector instance
    PXCFaceAnalysis::Detection *faceDetector=faceAnalyzer-
>DynamicCast<PXCFaceAnalysis::Detection>();
    if (!faceDetector) {
        wprintf_s(L"Failed to locate the face detection
functionality\n");
        return 3;
    }
}
```

```
    // Create landmark instance
    PXCFaceAnalysis::Landmark
*landmarkDetector=landmarkDetector=faceAnalyzer-
>DynamicCast<PXCFaceAnalysis::Landmark>();
    if (!landmarkDetector) {
        wprintf_s(L"Failed to locate the face landmark functionality\n");
        return 3;
    }

    // Set landmark profile
    PXCFaceAnalysis::Landmark::ProfileInfo lInfo={0};
    landmarkDetector->QueryProfile(0, &lInfo);
    landmarkDetector->SetProfile(&lInfo);

    // Create Renderer
    PXCSmartPtr<FaceRender> faceRender(new FaceRender(L"Landmark Detection
Sample"));

    int fnum;
    for (fnum=0;fnum<cmdl.m_nframes;fnum++) {
        PXCSmartArray<PXCImage> images;
        PXCSmartSPArray sps(2);

        /*** read and process frame */
        sts = capture.ReadStreamAsync(images, &sps[0]);
        if (sts<PXC_STATUS_NO_ERROR) break; // EOF

        sts = faceAnalyzer->ProcessImageAsync(images, &sps[1]);
        sts = sps.SynchronizeEx();
        if (sps[0]->Synchronize(0)<PXC_STATUS_NO_ERROR) break; // EOF

        // loop all faces
        faceRender->ClearData();
        for (int fidx = 0; ; fidx++) {
            pxcUID fid = 0;
            pxcU64 timeStamp = 0;
            sts = faceAnalyzer->QueryFace(fidx, &fid, &timeStamp);
            if (sts < PXC_STATUS_NO_ERROR) break; // no more faces

            // Query face detection results
            PXCFaceAnalysis::Detection::Data face_data;
            faceDetector->QueryData(fid, &face_data);
            faceRender->SetDetectionData(&face_data);

            // Query landmark points
            faceRender->SetLandmarkData(landmarkDetector, fid);
            faceRender->PrintLandmarkData(landmarkDetector, fid);
            wprintf_s(L"timestamp=%I64d, frame=%d\n", timeStamp, fnum);
        }

        if (!faceRender->RenderFrame(images[0])) break;
    }
    return 0;
}
```

Example 12: landmark detection sample code



Running the application will output the face detection window and landmarks shown in Figure 7. Note that the `FaceRender` utility is used to render the output frame with the face detection results. The functions `FaceRender::SetDetectionData` and `FaceRender::SetLandmarkData` are used to set the face detection/landmark data and the function `FaceRender::RenderFrame` is used to render each frame with the set face detection data.

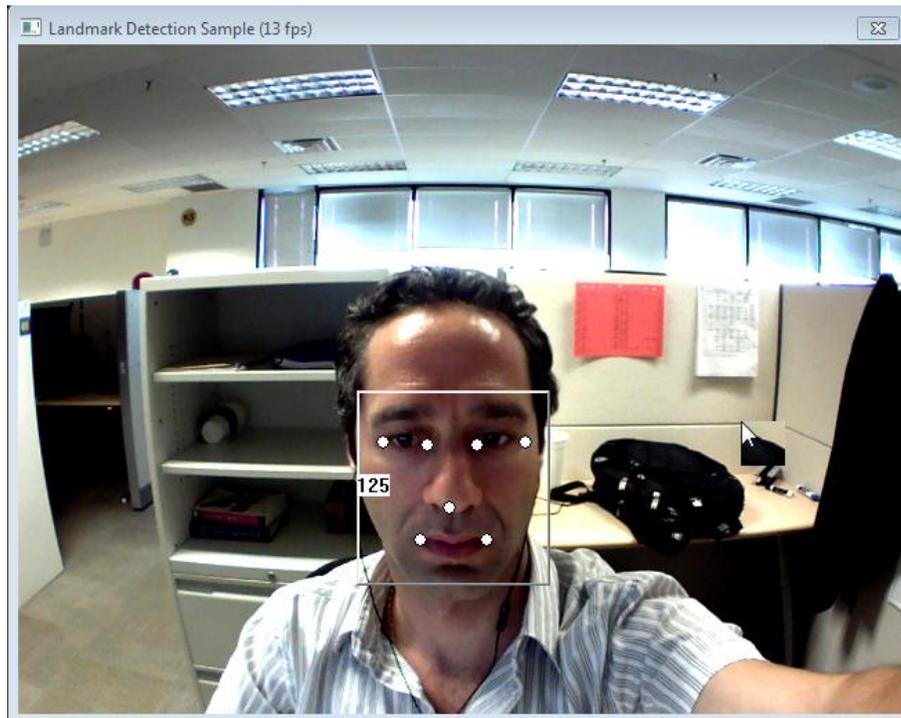


Figure 7: Landmark detection results