

# Intel® RealSense™ SDK 2014

## Hand Analysis Tutorial

With the Intel® RealSense™ SDK, you have access to robust, natural human-computer interaction (HCI) algorithms such as face tracking, finger tracking, gesture recognition, voice and speech recognition, fully textured 3D scanning and enhanced depth augmented reality.

With the SDK you can create Windows\* desktop applications that offer innovative user experiences.

This tutorial shows how to enable the hand analysis module of the SDK and enable its features, such as hand and finger tracking, robust 22-point skeletal joint tracking, and gesture recognition.

The module can also display event alert notifications in your application, and a basic rendering utility is provided to view the output data.

# Contents

- **Overview**
  - Hand Analysis Data
  - Gesture Tracking
  - Hand Alert Notification
- **Code Sample Files**
- **Creating a Session for Hand Analysis**
- **Hand Tracking**
  - Initialize the Pipeline
  - Stream Hand Analysis data
- **Gesture Tracking**
  - Initialize the Pipeline
  - Stream the Gesture Analysis Data
- **Alert Event Notifications**
  - Initialize the Pipeline
  - Stream the Alert Notification Data
- **Rendering the Frame**
- **Cleaning up the Pipeline**
- **Running the Code Sample**
- **To learn more**

# Overview

The Intel RealSense SDK supports input/output modules and algorithm modules. In this tutorial, you'll see how to use one of the algorithm modules in particular the hand analysis module. To learn more about I/O modules, see the [Capturing Raw Streams](#) tutorial.

The **PXCHandAnalysis** interface of the SDK is the representation of the hand analysis module. **PXCHandAnalysis** uses its class member functions to perform hand and joint tracking, gesture recognition, and alert event notifications for multiple hands in an image frame.

## Hand Analysis Data

Applications can locate and return a hand (or multiple hands) through the **PXCHandData** interface.

- Applications must capture hand data before they can capture any other data related to a hand.

The hand data is exposed to the application through the **iHand** interface that provides the following data:

**Mass Center**: Image and world coordinates of the calculated hand image mass center.

**Extremity Points**: Special tracking points such as the left most point and the right most point.

**Body Side**: Whether it is a left or right hand.

**Palm Orientation**: Estimation of where the hand is facing.

**Tracked Joints**: Position and rotations of the user's hand in world and image coordinates.

**Normalized Joints**: Posture of user's hand without changing dimensions of the hand, i.e., the distances between each joint (bone-length) are always the same.








**Finger Data**: Degree of foldedness and the radius of a particular fingertip.

**Segmentation Image**: Rectangle of the boundaries of only the hand in the image.

## Gesture Tracking

Using the **PXCHandData** interface, you can also extract gesture information. By default, certain

defined hand movements, also known as gesture packs, can be loaded for detection.

Gesture Name	Illustration	Description
<a href="#">spreadfingers</a>		The hand is open facing the camera with fingers pointing upwards.
<a href="#">fist</a>		Fingers folded into a fist.
<a href="#">tap</a>		Hand moving in z direction as if one is pressing a button.
<a href="#">thumb_down</a>		The hand is closed with the thumb finger pointing down.
<a href="#">thumb_up</a>		The hand is closed with the thumb finger pointing up.
<a href="#">two_fingers_pinch_open</a>		The thumb finger and the index finger touch each with vertical orientation of the hand.
<a href="#">v_sign</a>		Index and middle fingers extended in upwards direction.

## Hand Alert Notification

This module helps notify your application of certain useful information:

[Hand detection and tracking](#): Inform the application that a hand is identified and is tracked.

[Hand calibration](#): Inform the application that hand calibration data is available.

[Tracking Boundaries](#): Inform the application when the tracked hand goes out or is about to go out of the tracking boundaries.

# Code Sample Files

You can use either procedural calls or event callbacks to capture hand data, and relevant code samples showing both are listed in Table 1. Using event callbacks is usually preferred when developing console applications; procedural calls are often used for GUI applications. This tutorial's code samples use procedural calls.

The `handanalysis_render.cpp` file, provided with the code samples, contains the **HandRender** utility class that renders an image sample that contains hand analysis data. It is provided with this tutorial so that you do not have to create your own hand rendering algorithm.

Executable files (.exe) are provided in the Debug subfolder in the code sample directory.

Table 1: Hand Analysis Code Samples

Code Sample	For explanation, see:
Hand analysis using procedural calls Code sample file: <code>main_hand_analysis_procedural.cpp</code>	This Tutorial. Also see <a href="#">Hand Tracking using the SenseManager Procedural Functions</a> section of the SDK Reference Manual.
Hand analysis using event callbacks	<a href="#">Hand Tracking using the SenseManager Callback Functions</a> section of the SDK Reference Manual.
Gesture analysis using procedural calls or events	<a href="#">Gesture Recognition Data</a> section of the SDK Reference Manual.
Alert notification using procedural calls or events	<a href="#">Handle Alert Notification</a> section of the SDK Reference Manual.

# Creating a Session for Hand Analysis

1. Create a session and instance of the **PXCSenseManager** to add the functionality of the Intel RealSense SDK to your application.
2. Initialize an instance of the **HandRender** utility class so that you can render the captured image samples.

```
//SDK provided utility class used for rendering (packaged in libpxcutils.lib)
#include "handanalysis_render.h"

{
    // initialize the UtilRender instances
    HandRender *renderer = new HandRender(L"Procedural Hand Tracking");

    // create the PXCSenseManager
    PXCSenseManager *psm=0;
    psm = PXCSenseManager::CreateInstance();
    if (!psm) {
        wprintf_s(L"Unable to create the PXCSenseManager\n");
        return 1;
    }
}
}
```

The SDK core is represented by two interfaces:

- **PXCSession**: manages all of the modules of the SDK
- **PXCSenseManager**: organizes a pipeline by starting, stopping, and pausing the operations of its various modalities.

Note: Each session maintains its own pipeline that contains the I/O and algorithm modules.

# Hand Tracking

## Initialize the Pipeline

1. Enable the Depth Stream of given size using **EnableStream** and **PXCCapture** type.
2. Enable the hand analysis using **EnableHand**.
3. Initialize the pipeline using **Init**.
4. Retrieve hand module if ready using **QueryHand**.
5. Retrieve the hand data using **CreateOutput** on the hand module.
6. Create an image instance for the color sample.

Note: You must use sts of type **pxcStatus** for error checking.

```
// error checking Status
pxcStatus sts;

// select the color stream of size 640x480 and depth stream of size 640x480
psm->EnableStream(PXCCapture::STREAM_TYPE_DEPTH, 640, 480);

// enable hand analysis in the multimodal pipeline
sts = psm->EnableHand();
if (sts < PXC_STATUS_NO_ERROR) {
    wprintf_s(L"Unable to enable Hand Tracking\n");
    return 2;
}

// retrieve hand results if ready
PXCHandModule* handAnalyzer = psm->QueryHand();
if (!psm) {
    wprintf_s(L"Unable to retrieve hand results\n");
    return 2;
}

// initialize the PXC SenseManager
if(psm->Init() < PXC_STATUS_NO_ERROR) return 3;

PXCHandData* outputData = handAnalyzer->CreateOutput();

PXCIImage *colorIm = NULL;
```

## Stream Hand Analysis data

1. Acquire Frames in a loop using **AcquireFrame(true)**:
  - a. TRUE (aligned) to wait for color and depth samples to be ready in a given frame; else
  - b. FALSE (unaligned).
2. In every iteration of the frame loop first **Update()** on the hand module to update the Hand data.
3. Create data structures of type **JointData** to hold the data per frame.
4. Use **QueryHandData** to populate **iHand**, a hand with appropriate access order type.
5. On a given instance of **iHand** you can **QueryTrackedJoint** with **JointType** to extract individual joints.

```
// stream data
while (psm->AcquireFrame(true)>=PXC_STATUS_NO_ERROR)
{
    outputData->Update();

    // create data structs
    PXCHandData::JointData nodes[NUM_HANDS][PXCHandData::NUMBER_OF_JOINTS]={};

    // iterate through hands
    pxcUID handID;
    pxcU16 numOfHands = outputData->QueryNumberOfHands();
    for(pxcU16 i = 0 ; i < numOfHands ; i++)
    {
        // get hand joints by time of appearance
        PXCHandData::IHand* handData;
        if(outputData->QueryHandData(PXCHandData::ACCESS_ORDER_BY_TIME,i,handData) ==
        PXC_STATUS_NO_ERROR)
        {
            // iterate through Joints
            for(int j = 0; j < PXCHandData::NUMBER_OF_JOINTS ; j++)
            {
                handData->QueryTrackedJoint((PXCHandData::JointType)),nodes[i][j]);
            }
        }
    }
}
```



# Gesture Tracking

## Initialize the Pipeline

1. Enable the Depth Stream of given size using **EnableStream** and **PXCCapture** type.
2. Enable the hand analysis using **EnableHand** and Initialize the pipeline using **Init**.
3. Retrieve hand module if ready using **QueryHand**.
4. Use **PXCHandConfiguration** to configure and enable all gestures
5. **Apply** the changes.

```
// select the depth stream of size 640x480
psm->EnableStream(PXCCapture::STREAM_TYPE_DEPTH, 640, 480);

// enable hand analysis in the multimodal pipeline
pxcStatus sts = psm->EnableHand();
if (sts < PXC_STATUS_NO_ERROR) {
    wprintf_s(L"Unable to enable Hand Tracking\n");
    return 2;
}

// retrieve hand results if ready
PXCHandModule* handAnalyzer = psm->QueryHand();
if (!psm) {
    wprintf_s(L"Unable to retrieve hand results\n");
    return 2;
}

// initialize the PXC SenseManager
if (psm->Init() < PXC_STATUS_NO_ERROR) return 3;

PXCHandData* outputData = handAnalyzer->CreateOutput();

PXCHandConfiguration* config = handAnalyzer->CreateActiveConfiguration();
config->EnableAllGestures();
config->ApplyChanges();
```

## Stream the Gesture Analysis Data

1. Acquire Frames in a loop using **AcquireFrame(true)**.
  2. In every iteration of the frame loop first **Update()** on the hand module to update the Hand data.
  3. Iterate through fired gestures using **QueryFiredGesturesNumber** and populate the gesture data using **QueryFiredGestureData**.
  4. You can access hand data like **QueryBodySide** (left or right hand) related to the fired gesture using **QueryHandDataById** and **gestureData.handId**.
- The gesture is saved using **wmemcpy\_s()**. Otherwise, gesture data is not saved for current gesture.

```
// stream data
while (psm->AcquireFrame(true)>=PXC_STATUS_NO_ERROR)
{
    outputData->Update();
    // create data structs
    PXCHandData::GestureData gestureData;
    pxcCHAR gestures[NUM_HANDS][PXCHandData::MAX_NAME_SIZE] = {};
    PXCHandData::BodySideType handSide[NUM_HANDS] = {PXCHandData::BODY_SIDE_UNKNOWN};

    // iterate through fired gestures
    for(unsigned int i = 0; i < outputData->QueryFiredGesturesNumber(); i++)
    {
        // initialize data
        wmemset(gestures[i], 0, sizeof(gestures[i]));
        handSide[i] = PXCHandData::BODY_SIDE_UNKNOWN;
        // get fired gesture data
        if(outputData->QueryFiredGestureData(i,gestureData) == PXC_STATUS_NO_ERROR)
        {
            // get hand data related to fired gesture
            PXCHandData::IHand* handData;
            if(outputData->QueryHandDataById(gestureData.handId,handData) ==
                PXC_STATUS_NO_ERROR)
            {
                // save gesture only if you know that its right/left hand
                if(!handData->QueryBodySide() ==
                    PXCHandData::BODY_SIDE_UNKNOWN)
                {
                    memcpy_s (gestures[i],PXCHandData::MAX_NAME_SIZE*2,
                        gestureData.name, sizeof(gestureData.name));
                    handSide[i] = handData->QueryBodySide();
                }
            }
        }
    }
}
}
```

# Alert Event Notifications

## Initialize the Pipeline

1. Enable the Depth Stream of given size using **EnableStream** and **PXCCapture** type.
2. Enable the hand analysis using **EnableHand** and Initialize the pipeline using **Init**.
3. Retrieve hand module if ready using **QueryHand**.
4. Use **PXCHandConfiguration** to configure and **EnableAllAlerts**.
5. **Apply** the changes.

```
// select the color stream of size 640x480 and depth stream of size 320x240
psm->EnableStream(PXCCapture::STREAM_TYPE_DEPTH, 640, 480);

// enable hand analysis in the multimodal pipeline
pxcStatus sts = psm->EnableHand();
if (sts < PXC_STATUS_NO_ERROR) {
    wprintf_s(L"Unable to enable Hand Tracking\n");
    return 2;
}

// retrieve hand results if ready
PXCHandModule* handAnalyzer = psm->QueryHand();
if (!psm) {
    wprintf_s(L"Unable to retrieve hand results\n");
    return 2;
}

// initialize the PXCSeSenseManager
if(psm->Init() < PXC_STATUS_NO_ERROR) return 3;

PXCHandData* outputData = handAnalyzer->CreateOutput();

PXCHandConfiguration* config = handAnalyzer->CreateActiveConfiguration();
config->EnableAllAlerts();
config->ApplyChanges();
```

## Stream the Alert Notification Data

1. Acquire Frames in a loop using **AcquireFrame(true)**.
2. In every iteration of the frame loop first **Update()** on the hand module to update the Hand data.
3. Iterate through fired alerts using **QueryFiredAlertsNumber** and populate the alert data using **QueryFiredAlertData**.
4. Using **alertData.label** you can compare in a switch statement all available alerts under the **PXCFaceData** alert type.

```
// stream data
while (psm->AcquireFrame(true)>=PXC_STATUS_NO_ERROR)
{
    outputData->Update();
    // iterate through Alerts
    PXCHandData::AlertData alertData;
    for(int i = 0 ; i <outputData->QueryFiredAlertsNumber() ; ++i)
    {
        if(outputData->QueryFiredAlertData(i,alertData)==PXC_STATUS_NO_ERROR)
        {
            // Display last alert - see AlertData::Label for all available alerts
            switch(alertData.label)
            {
                case PXCFaceData::AlertData:: ALERT_NEW_FACE_DETECTED:
                {
                    wprintf_s(L"Last Alert: Face Detected\n", alertData.faceId);
                    break;
                }
                case PXCFaceData::AlertData:: ALERT_FACE_LOST:
                {
                    wprintf_s(L"Last Alert: Face Not Detected\n", alertData.faceId);
                    break;
                }
            }
        }
    }
}
```

# Rendering the Frame

1. Retrieve a sample using **QuerySample**.
  2. Retrieve the specific image or frame type from the sample, by saving it in a variable of type **PXCImage**.
- Make sure to enable all the possible required streams in the **initializing pipeline phase**, if not enabled the sample types will **return null**.
  - Use the **RenderFrame** method provided in the custom renderer to render all the above set algorithms.

```
//while loop per frame
{
    // retrieve all available image samples
    PXCapture::Sample *sample = psm->QuerySample();

    // retrieve the image or frame by type
    colorIm = sample->depth;

    // render the frame
    if (!renderer->RenderFrame(colorIm)) break;
}
```

# Cleaning up the Pipeline

1. Make sure to **ReleaseFrame** in every iteration of the while loop, so that you can acquire a fresh frame in the next iteration.
2. Make sure to **Release** the renderer at the end to delete the **FaceRenderer** instance.
3. If used, also release any instance of **PXCHandConfiguration** using **Release** on it.
4. It is very important to close the last opened session, in this case the instance of **PXCStateManager** by using **Release** to indicate that the camera can stop streaming.

```
//In Main
{
    //in the while loop per frame
    {
        // release or unlock the current frame to go fetch the next frame
        psm->ReleaseFrame();
    }
    //End of while loop per frame

    // delete the HandRender instance
    renderer->Release();

    // close the Hand Configuration instance
    config->Release();

    // close the last opened stream and release any session
    //and processing module instances
    psm->Release();

    return 0;
}
//End of Main
```

# Running the Code Sample

You can run the code sample two ways:

1. Build and Run the 3\_Hands\_Analysis sample in Visual Studio\*.
2. Run the executables in the “Debug” subfolder of the code sample directory.

Figure 1 shows a rendered depth image frame that displays the hand tracking data as the 22 tracked joints, Alert data, and Gesture Data along with the Body side of each gesture.

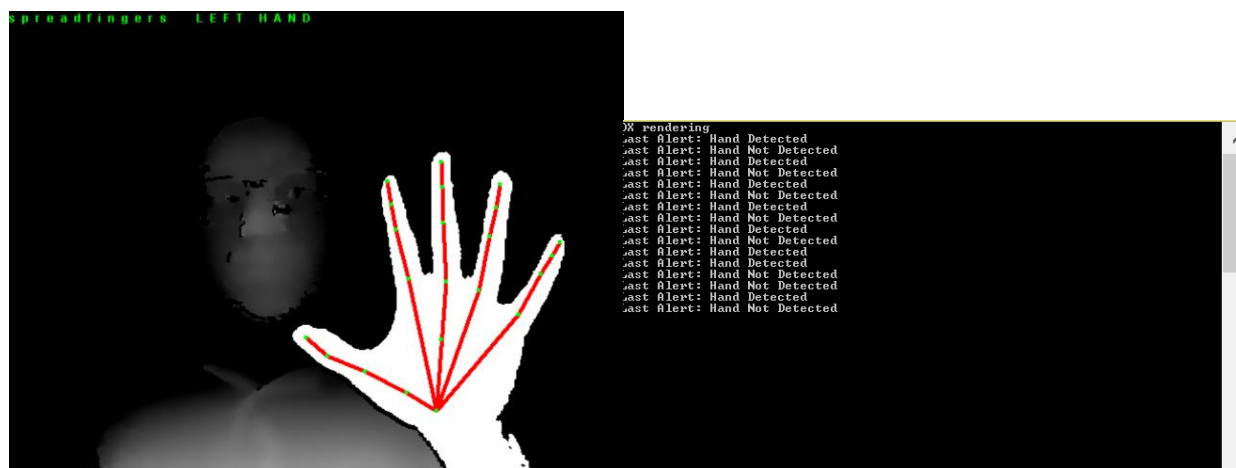


Figure 1. Rendered Depth Sample with Hand Tracking data, Alert data, and Gesture data

## To learn more

- The [Intel RealSense SDK Reference Manual](#) is your complete reference guide and contains API definitions, advanced programming techniques, frameworks, and other need-to-know topics.
- The Intel RealSense SDK allows the hand analysis module to adapt to a user's hand over time. Check out how to save **hand calibration data** for specific users in the [Hand Calibration Data](#) section of the SDK Reference Manual.