Intel® RealSense™ SDK

Speech Recognition and Synthesis 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, speech recognition and synthesis, fully textured 3D scanning and enhanced depth augmented reality.

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

In this tutorial, you'll learn how to use the SDK speech modules that provide command and control, dictation, and text-to-speech capabilities.

Contents

Overview

The SDK speech modules provide command and control, dictation, and text-to-speech capabilities.

Speech Recognition

Translates input speech into text. The two modes of operations are:

Command and Control

The application defines a list of context words as the command list, and the SDK module recognizes speech based only on the command list.

Dictation

The SDK module returns the closest match it can to dictated sentence.

Speech Synthesis

Text to speech, also called speech synthesis, translates text back to speech.

Table 1: Code Samples

Code Sample	For more information, see:
Dictation using event callbacks File: main_speech_recognition.cpp	This Tutorial. Also see Command Control and Dictation section in the SDK Reference Manual.
Command and Control	Configure the Command and Control List sections in the SDK Reference Manual.
Speech synthesis using event callbacks File: main_speech_synthesis.cpp	This Tutorial. Also see Speech Synthesis section in the SDK Reference Manual.

Creating a Session

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.

The first step when creating an application that uses the Intel RealSense SDK is to create a session. A session can be created explicitly by creating an instance of **PXCSession**

```
 #include <pxcsensemanager.h>
#include <pxcspeechrecognition.h>
{
        // Create and retrieve a session
        PXCSession *session = PXCSession::CreateInstance();
        if (session == NULL) {
                wprintf_s(L"Session not created by PXCSession\n");
                return 1;
        }

}
```

Speech Recognition

Initializing the Pipeline

1. Retrieve an instance of **PXCSpeechRecognition** from the active session using **CreateImpl<PXCSpeechRecognition>**.
2. Initialize the module by querying a speech recognition profile using **QueryProfile**. You can manipulate components like the length of silence at the end of a sentence, supported language, and recognition confidence threshold.
3. After making the required changes, set the profile using **SetProfile**.
4. Set the kind of recognition mode using **SetDictation**.

```
//Create an instance of the PXCSpeechRecognition
PXCSpeechRecognition *sr=0;
sts = session->CreateImpl<PXCSpeechRecognition>(&sr);
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
            wprintf_s(L"Failed to create an instance of the PXCSpeechRecognition\n");
            return 2;
  }

//Initialize the Module
PXCSpeechRecognition::ProfileInfo pinfo;
sr->QueryProfile(0,&pinfo);
sts = sr->SetProfile(&pinfo);
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
            wprintf_s(L"Failed to Configure the Module\n");
            return 3;
  }

//Set the Recognition mode
//command and control mode or dictation mode
sts = sr->SetDictation();
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
           wprintf_s(L"Failed to Set the Recognition mode \n");
           return 4;
  }
```

You can set up a Configure the Command and Control List by supplying an array of recognizable words to **BuildGrammarFromStringList** and then setting the kind of recognition using **SetGrammar**.

Setting up an Event Handler

The application receives events on recognition activities.

1. The module sends **OnRecognition** events along with **RecognitionData** when there are active recognition results.
2. The module sends **OnAlert** events along with **AlertData** when there are warnings or errors such as volume too high or too low.

```
class MyHandler: public PXCSpeechRecognition::Handler {
public:
  virtual void PXCAPI OnRecognition(const PXCSpeechRecognition::RecognitionData *data) {
                    wprintf_s(L"Output: %s\n",data->scores[0].sentence);
  }

  virtual void PXCAPI OnAlert(const PXCSpeechRecognition::AlertData *data) {
            if(data->label == PXCSpeechRecognition::ALERT_SPEECH_BEGIN)
                    wprintf_s(L"Alert: SPEECH_BEGIN\n");
            else if (data->label == PXCSpeechRecognition::ALERT_SPEECH_END)
                     wprintf_s(L"Alert: SPEECH_END\n");
  }
};
MyHandler handler; // handler for PXCSpeechRecognition
```

Start/Stop Speech Recognition

1. Start speech recognition using **StartRec** by supplying the appropriate speech recognition handler.
2. Stop speech recognition using **StopRec**, which stops any active speech recognition event handler.

```
//Start the speech recognition with the event handler
sts = sr->StartRec(NULL, &handler);
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Failed to Start the handler \n");
        return 5;
}

while(true) { if (GetAsyncKeyState(VK_ESCAPE)) break; }  //looping infinitely until escape is pressed

//Stop the event handler that handles the speech recognition
sts = sr->StopRec();
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
        wprintf_s(L"Failed to Stop the handler \n");
        return 6;
 }
```

Speech Synthesis

Initializing the Pipeline

1. Retrieve an instance of **PXCSpeechSynthesis** from the active session using **CreateImpl< PXCSpeechSynthesis>**.
2. Initialize the module by querying a speech synthesis profile using **QueryProfile**. You can manipulate components like the speaking speed, volume, pitch, end of sentence wait duration, and the supported language.
3. After making the required changes, set the profile using **SetProfile**.
4. Initialize the **audio renderer utility** provided to render audio by supplying profile information.

```
//Create an instance of the PXCSpeechSynthesis
PXCSpeechSynthesis *tts=0;
sts = session->CreateImpl<PXCSpeechSynthesis>(&tts);
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
            wprintf_s(L"Failed to create an instance of the PXCSpeechSynthesis\n");
             return 2;
  }

//Initialize the Module
PXCSpeechSynthesis::ProfileInfo pinfo;
tts->QueryProfile(0,&pinfo);
sts = tts->SetProfile(&pinfo);
if (sts != pxcStatus::PXC_STATUS_NO_ERROR) {
             wprintf_s(L"Failed to Initialize the Module\n");
             return 3;
  }

//initialize the renderer with Synthesis profile
AudioRenderer renderer(&pinfo);
```

Retrieving and Rendering the Audio from Text

1. Use **BuildSentence** to synthesize a sentence. The synthesized speech is stored as multiple audio buffers identified by the sentence identifier (non-zero unique value).
2. Use **QueryBuffer** to retrieve the synthesized speech buffers.
3. Pass the audio buffer to the audio renderer utility class using **RenderAudio** to playback.
4. Use **ReleaseSentence** to release any resources allocated for the synthesized speech.

```
//String to input from console
string text;

//Loop through to repeat
while(true){

        cout<<"\nEnter the Text and hit Enter\n";

        pxcCHAR sentence[256];
        getline(cin,text);
        wstring widestr = std::wstring(text.begin(), text.end());
        wcscpy_s(sentence,widestr.c_str());

        // Synthesize the text string
        tts->BuildSentence(1,sentence);

        // Retrieve the synthesized speech
        int nbuffers=tts->QueryBufferNum(1);
        for (int i=0;i<nbuffers;i++) {

         PXCAudio *audio=tts->QueryBuffer(1, i);
                // send audio to the audio output device
                renderer.RenderAudio(audio);
        }

        // Clean up
        tts->ReleaseSentence(1);
}
```

Cleaning Up the Pipeline

After your application is done using the module, you must “clean up”. Release the active session and processing module instance using **Release()** on the **PXCSession** instance.

```
//Release the session instance
session->Release();
```

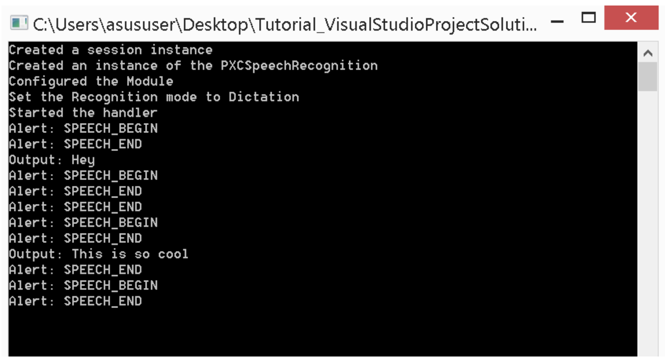
Now you have all the information to use speech recognition and speech synthesis modules provided in the Intel RealSense SDK.

Running the Code Samples

You can run these code samples two ways:

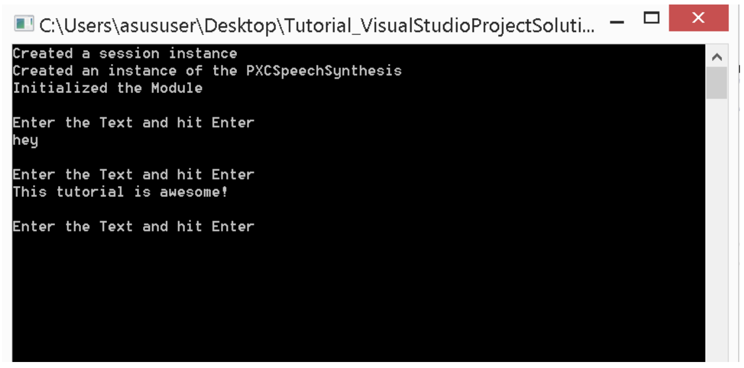
1. Build and run the **Speech_Recognition** or **Speech_Synthesis** sample in Visual Studio*.
2. Run the executables found in the “Release” subfolder in the tutorial code sample directory.

Figures 1 and 2 show the console output while running speech recognition, speech synthesis.


C:\Users\asususer\Desktop\Tutorial_VisualStudioProjectSoluti...

```
Created a session instance
Created an instance of the PXCSpeechRecognition
Configured the Module
Set the Recognition mode to Dictation
Started the handler
Alert: SPEECH_BEGIN
Alert: SPEECH_END
Output: Hey
Alert: SPEECH_BEGIN
Alert: SPEECH_END
Alert: SPEECH_END
Alert: SPEECH_BEGIN
Alert: SPEECH_END
Output: This is so cool
Alert: SPEECH_END
Alert: SPEECH_BEGIN
Alert: SPEECH_END
```

Figure 1. Speech Recognition


C:\Users\asususer\Desktop\Tutorial_VisualStudioProjectSoluti...

```
Created a session instance
Created an instance of the PXCSpeechSynthesis
Initialized the Module

Enter the Text and hit Enter
hey

Enter the Text and hit Enter
This tutorial is awesome!

Enter the Text and hit Enter
```

Figure 2. Speech Synthesis

To learn more

- The SDK Reference Manual is your complete reference guide and contains API definitions, advanced programming techniques, frameworks, and other need-to-know topics.