Getting started with the Camera APIs for native code

This tutorial explains how to use the native camera interfaces in the Windows Phone 8 SDK.

Introduction

The camera APIs available on Windows Phone 8 provides both high-, mid- and low-level scenarios. High-level use can be in the form of using an `AudioVideoCaptureDevice` instance to record video to a file, or taking a sequence of photos using `PhotoCaptureDevice`. Mid-level usage can be in the form of using the ability to read out the preview buffer from an active `AudioVideoCaptureDevice`, using the method `AudioVideoCaptureDevice.GetPreviewBufferArgb` or one of the alternatives. Low-level access requires the use of the native interfaces introduced in Windows Phone 8, and in this article I will show how to achieve that.

Developing using the Camera APIs for native code

The native interfaces, as described on Camera APIs for native code (Windows Phone 8), are all that is needed to implement native access to the advanced camera APIs. Some of the interfaces are implemented by Windows Phone 8, and some of the interfaces must be implemented by the application.

The interfaces implemented by the OS are `ICameraCaptureDeviceNative` and `IAudioVideoCaptureDeviceNative`, and the application-implemented interfaces are `ICameraCapturePreviewSink` and `ICameraCaptureSampleSink`.

To retrieve an instance of one of the OS implemented interfaces, the procedure starts with either an instance of `AudioVideoCaptureDevice` or `PhotoCaptureDevice`.

An example of how this is done, can be seen in the following example:

```
AudioVideoCaptureDevice captureDevice;

// ...

// Retrieve the native ICameraCaptureDeviceNative interface from the managed video
capture device
ICameraCaptureDeviceNative *iCameraCaptureDeviceNative = NULL;
HRESULT hr =
reinterpret_cast<IUnknown*>(captureDevice)->QueryInterface(__uuidof(ICameraCaptureDeviceNative),
(void**) &iCameraCaptureDeviceNative);
```

The `ICameraCaptureDeviceNative` interface is of interest to both photo- and audio/video-applications, as it provides access to a number of interesting methods. The first method of interest is `SetPreviewSink`. This method takes a pointer to an instance of the `ICameraCapturePreviewSink` interface. This means that the application will receive the preview frames and can process them in any way wanted. Related to this method is `SetPreviewFormat`, which sets the format of the preview-frames delivered (i.e. ARGB, NV12, etc.). The other two methods in this interface, `SetDevice` (used to set the DirectX 11 device and context) and `GetPreviewBufferTexture` (used to fill a DirectX 11 texture with the preview frames). Please note that the DirectX-related part of the interface is not demonstrated in this example.

The `IAudioVideoCaptureDeviceNative`-interface provides access to methods geared towards video- and audio-applications. The method `SetAudioSampleSink` sets the sink to receive audio samples, and the method `SetVideoSampleSink` sets the sink to receive video samples. Please note that both methods take a pointer to an instance of the `ICameraCaptureSampleSink` interface and that the `ICameraCaptureSampleSink` interface itself does not contain any way to identify whether the sample received is audio or video, so therefore I would recommend to implement it as two seperate sinks, one to handle audio and the other to handle video samples.

Limitations on capturing and preview dimensions

The following table shows which combinations of capture and preview resolutions that works in practice (this has been tested on a Lumia 920 device).

Type	Aspect ratio	Capturing resolution	Preview resolution
Photo	4:3	3264x2448, 2592x1936, 2048x1536, 640x480	1024x768, 640x480

Photo	16:9	3552x2000, 2592x1456	1280x720, 800x448
Video	4:3	640x480, 320x240, 160x120	1024x768, 640x480
Video	16:9	1920x1080, 1280x720	1280x720, 800x448

Implementing the interfaces

To implement the interfaces, the following files must be included

```
#include <implements.h>
#include <Windows.Phone.Media.Capture.h>
#include <Windows.Phone.Media.Capture.Native.h> // This file contains the interface
definitions for the native interfaces
```

```
class CameraCapturePreviewSink :
 public Microsoft::WRL::RuntimeClass<
  Microsoft::WRL::RuntimeClassFlags<Microsoft::WRL::RuntimeClassType::ClassicCom>,
  ICameraCapturePreviewSink>
{
 IFACEMETHODIMP_(void) OnFrameAvailable(
  DXGI_FORMAT format,
  UINT width,
  UINT height,
  BYTE* pixels);
};
```

```
class CameraCaptureSampleSink :
 public Microsoft::WRL::RuntimeClass<
  Microsoft::WRL::RuntimeClassFlags<Microsoft::WRL::RuntimeClassType::ClassicCom>,
  ICameraCaptureSampleSink>
{
 IFACEMETHODIMP_(void) OnSampleAvailable(
  ULONGLONG hnsPresentationTime,
  ULONGLONG hnsSampleDuration,
  DWORD cbSample,
  BYTE* pSample);
};
```

Use cases

Native access to the camera and audio data is highly sought after in a number of fields

- Computer vision (as the algorithms are usually heavy on CPU and memory bandwidth)
- VoIP (as you can get access to the video and audio the moment they have been either sampled or encoded by hardware/OS)

My own idea for this project was as starting point for use in a computer vision project. This would include use of the SURF algorithm, which is an obvious candidate for implementing in native code, as both the summed area table generation and further parts of the algorithm can benefit from SIMD processing. Furthermore, access to the camera data without the overhead of the managed runtime, would benefit the speed of the implementation greatly, as there would be no unnecessary transfers between native and managed code of large amounts of data and the ability to optimize using NEON instructions where applicable.

Sample code

Please see this attached project for a full implementation of the interfaces mentioned in this article File:NativeCameraInterface.zip.