



## CDR Programmer's Guide: Using USBCam2 / USBCam4

Schick Technologies, Inc.  
30-30 47<sup>th</sup> Avenue  
Long Island City, NY 11101  
USA

(718) 937-5765  
(718) 937-5962 (fax)

EU Authorized Representative  
Sirona Dental GmbH  
General Counsel Europe /Asia Pacific  
Wasserfeldstraße 30  
A - 5020 Salzburg, Austria

Copyright © 2011 by Schick Technologies, Inc.  
All Rights Reserved

CDR is a registered trademark of Schick Technologies, Inc. Schick Technologies, Inc. products are covered by one or more of the following US Patents.

Wired CDR® Sensors US 5,434,418; US 6,069,935;  
US 6,134,298

Wired and Wireless CDR® Sensors (Both) US 5,179,579; US 5,912,942; US 5,841,126; US 6,456,326; US 6,549,235; US 6,570,617; US 6,744,068; US 7,369,166

CDR® Wireless Sensors US 5,514,873; US 6,924,486;  
US 6,972,411; US 7,072,443; US 7,171,181; US 7,193,219; US D493,892; US 7,090,395

USBCAM® US 6,002,424; US 5,908,294

Positioning Systems US 6,811,312

SDX US 7,551,720

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and Schick Technologies, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

May 4, 2011



Printed in the United States of America

This document was originally prepared in English

# Contents

<b>1. Overview .....</b>	<b>1</b>
1.1 The Role of WDM .....	1
1.2 DirectShow Architecture.....	1
1.3 Filter Graphs .....	1
1.4 DirectShow Objects .....	2
1.4.1 Device Object .....	2
1.4.2 Filter Graph Object.....	2
1.4.3 Renderer Object .....	2
1.4.4 Sample Grabber Object.....	2
1.4.5 Manager Object.....	2
1.5 Relations among DirectShow Objects .....	2
<b>2. Designing a FilterGraph for the USBCam.....</b>	<b>3</b>
2.1 Simple FilterGraph.....	3
2.2 FilterGraph with SmartTee Preview .....	3
<b>3. Using USBCam Handpiece Button .....</b>	<b>4</b>
3.1 Polling Frequency .....	4
3.2 Checking Button Status.....	4
3.3 Camera Object Overview .....	4
3.3.1 Camera Object Properties .....	4
3.3.2 Camera Object Methods.....	5

# Introduction for Developers

This document provides software developers with information needed to connect their applications to the USBCam2 and USBCam4 intraoral cameras. Details on programming the camera handpiece capture button and accessing other options are provided by this document and the code samples supplied with it.

## Differences between USBCam2 and USBCam4

USBCam2 drivers are compatible with the 32-bit systems of Windows XP and Vista with Service Pack 1. USBCam4 drivers are compatible with 32- and 64-bits systems of Windows 7, Windows XP and Vista, are digitally signed, and are Schick-branded to avoid conflicts with other dental cameras or commercially available consumer products.

Differences in friendly\_name will help software developers distinguish the USBCam2 and USBCam4 cameras, as indicated below:

- USBCam2 – Schick Technologies USBCam2
- USBCam4 – Schick Technologies USBCam4

## Release Versions

The table below describes the releases related to this SDK.

SDK Document Date	Current with . . .	Driver Release Date
May 2011	USBCam4 Driver (4.6.4.318)	May 2011
May 2006	USBCam2 Driver (4.1.1.50)	February 2007

Please note that some options and implementations described in this SDK are specific to the software releases that comprise them. If a function is unavailable, or is not working properly with the camera hardware, it may not be supported by a specific software release.

## For More Information

If you have questions related to this SDK, please post them to our on-line bulletin board service.

<http://www.schicktech.com/bbs3/>

# 1. Overview

The DirectShow framework must be used to develop applications for the USBCam2 and USBCam4. This section provides a brief overview of this technology. For more information, refer to the following link, which can be found on Microsoft's MSDN website:

[http://msdn.microsoft.com/en-us/library/dd375454\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd375454(VS.85).aspx)

## 1.1 The Role of WDM

DirectShow works through WDM streaming on the kernel level. WDM Streaming provides the means for synchronizing the rendering of large volumes of data in real time. It works by moving most, if not all, data streaming tasks down from user-mode to kernel-mode drivers or onto hardware. Transitions to user-mode are kept to a minimum.

## 1.2 DirectShow Architecture

DirectShow is both a set of COM interfaces and a library of objects that implement those interfaces. For example, the framework provides a standard video renderer, a COM server, implementing a number of DirectShow interfaces.

## 1.3 Filter Graphs

The DirectShow architecture defines how to control and process multimedia streams by using modular components called filters. A filter is a COM object with “pins” allowing connections to other filters. The pins are COM interfaces, accepting pointers to other pin interfaces, and thereby connecting them. A filter may operate on streaming data in any number of ways, retrieving, processing, or transforming it before sending it to the next filter. A group of filters connected in such a manner is called a filter graph. An object called the filter graph manager controls how the filter graph is assembled and how data is moved through the graph. An application deals with the graph through the graph manager, using methods on manager interfaces to configure the graph, to stream data through its filters, and to determine whether the destination is a display window or an AVI file (or both).

Many kinds of graphs can be constructed for different purposes. Two common graphs are preview and capture. Preview graphs are used to stream data from a capture card to a video display window. Capture graphs take in the name of an AVI file and stream to that file on the hard drive. Capture graphs may support preview-style streaming to the display window during capture – this is hardware dependent. If such previewing is allowed, the capture graph will be constructed using preview pins for display rendering. There's some overlap in the other direction too – for example, preview graphs use capture pins to configure the data stream (as in frame rate.)

## 1.4 DirectShow Objects

There are several major COM objects used in a typical DirectShow application.

### 1.4.1 Device Object

This object represents the hardware device and implements the IBaseFilter interface. Device Objects always have at least one output capture pin and possibly one or more additional capture or preview pins. Device Objects have no input pins. Device objects implement the IAMStreamConfig interface for configuring the video stream through a particular pin..

### 1.4.2 Filter Graph Object

This object represents a filter graph and implements the IGraphBuilder interface. This object also implements the IMediaControl interface to run, pause and stop the video stream.

### 1.4.3 Renderer Object

This object renders the video to the destination window or file. It generally implements the IVideoWindow interface for display. Renderer objects always have a single input pin.

### 1.4.4 Sample Grabber Object

This object was introduced in Microsoft DirectX 8.0. It can be placed in a capture stream prior to a renderer. It is used to sample frames before they are captured. In this method a still frame may be “grabbed” at any point in the video stream.

### 1.4.5 Manager Object

The final object is the graph manager and implements the ICaptureGraphBuilder2 interface. This object is used to initialize the stream rendering, to obtain interface pointers supported by the Device Object, and to control the capturing of video to an AVI file.

## 1.5 Relations among DirectShow Objects

The following pseudo-code shows how the objects establish their relations to each other:

```
IBaseFilter *deviceFilter;  
IGraphBuilder *graphBuilder;  
ICaptureGraphBuilder *captureGraphBuilder;  
  
//Initilialize objects though calls to CoCreateInstance() (not shown)  
  
//Set the filter graph controlled by the filter graph manager  
captureGraphBuilder->SetFilterGraph(graphBuilder) ;  
  
//Add the device object as a filter to the filter graph. This is enough for most basic  
needs  
graphBuilder->AddFilter(deviceFilter);
```

With the object relations in place, we are ready to preview or capture video.

## 2. Designing a FilterGraph for the USBCam

The USBCam filter device contains two output pins: a capture pin and a still pin. We recommend using the capture pin for live video display and still capture. The device can be configured in any acceptable filter graph arrangement; however, we have found that using a Sample Grabber (**DirectX 9.0 or higher**) provides the best performance and reliability for grabbing still frames. See the DirectX SDK for details in using the Sample Grabber.

### 2.1 Simple FilterGraph

For most systems, the filter graph can be configured by simply inserting a Sample Grabber between the capture pin and the video renderer. See **Figure 1** for a sample of this graph. The perceived frame rate is dependent on the capabilities on the CPU and video hardware. While this is typically not a problem for modern systems (PIII, 800MHz or higher), on some older machines, this filter may produce an undesirable frame rate.

### 2.2 FilterGraph with SmartTee Preview

**Figure 2** provides an alternative filter graph which uses a Smart Tee to split the capture and preview streams. The preview stream is sent directly to the video renderer, while the Sample Grabber is placed in the capture stream. This configuration can greatly improve performance on slower systems, but is more complex to configure in code.

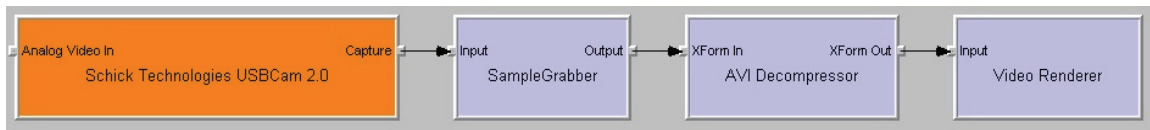


Figure 1. Simple USBCam FilterGraph

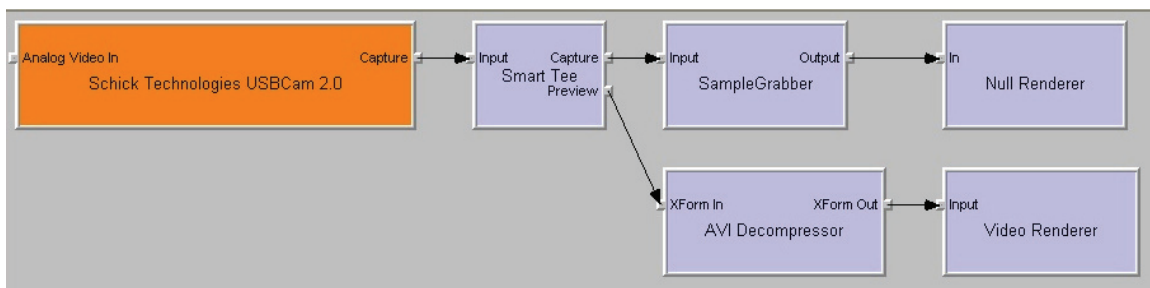


Figure 2. FilterGraph with SmartTee Preview

## 3. Using USBCam Handpiece Button

A button on the USBCam can be configured in software to perform a desired task, such as, toggling the freeze/live video stream or capturing a single frame. In order to make use of this button, the application must continuously poll a custom property of the USBCam filter for the push-button status. When the button is pressed, the returned value will change and the application can take appropriate action.

### 3.1 Polling Frequency

We recommend a polling frequency of 500ms. Although, the device can be polled more frequently, it may be possible to receive two state transitions for one push, so care must be taken to properly debounce the signal in software.

### 3.2 Checking Button Status

The button status is monitored by using the `IsButtonPressed` method, which returns a Boolean value of `True` if pressed, or `False` for any other condition. An example of how this operation is accomplished in code can be found in the C++ project, `SampleButtonDlg.cpp`, supplied with the SDK.

### 3.3 Camera Object Overview

Class GUID: {BDBABB31-5B48-4E75-8B3A-2028AE335304}

Program ID: Instantiation by Program ID is not guaranteed to be successful

#### 3.3.1 Camera Object Properties

##### 3.3.1.1 `CCameraClass.ShutterMode` Property

Member of the ICamera interface of the USBCam20SDK library.

DATA TYPE  
Object

ACCESS  
Read / Write

DESCRIPTION  
Returns a value associated with setting the camera's shutter speed



### **3.3.1.2 CCameraClass.WhiteBalanceProperty**

Member of the ICamera interface of the USBCam20SDK library.

DATA TYPE

Object

ACCESS

Read / Write

DESCRIPTION

Returns a value associated with features in white balance mode (WBM)

### **3.3.2 Camera Object Methods**

#### **3.3.2.1 Method CCameraClass.IsButtonPressed**

Member of the ICamera interface of the USBCam20SDK library.

SYNTAX

Method CCameraClass.IsButtonPressed( )

RETURNS

Boolean -- True if the capture button is pressed; otherwise, False

PARAMETERS

None

DESCRIPTION

Returns the status of the snapshot capture button