**HOW DO I …**

Use a live camera feed in my application?

Greetings. In this video we will see how to create an application that makes use of the new live camera feeds available in Windows Phone Mango. Not only that, but we will also show how to take the live camera feed and apply an effect to it by processing the feed’s data before displaying it.

# Live camera feeds

Windows Phone Mango offers two possible approaches for accessing a live camera feed

One approach is using Silverlight 4’s Webcam feature. This allows to record audio and video, with data from the camera supplied to the application via an asynchronous callback.

A different approach is to use the PhotoCamera class. This class allows us to control the phone’s camera and respond to events such as the camera’s button being pressed, allowing for better integration with the device. When using the phone camera, we need to actively request data from the camera from our code.

We will be using the PhotoCamera as it gives us greater control and ease of use in our scenario – to create an application that allows taking pictures after applying various effects to them.

# Demo

Let us have a look at a nearly finished version of the application, with which we will demonstrate how easy it is to add and manipulate a live camera feed.

1. Start by going over the solution in the starter folder. The main page, used for displaying the live feed and applying effects (focus on the rectangle containing the video brush), the preview page used for showing captured images and the various effect classes which process images to apply an effect.
2. Run the application and show its current state (without a camera feed).
3. Open **MainPage.xaml** and locate the radio button called “effectsRadioButton”. Add the following event handler: Click="RadioButton\_Click".
4. Locate the radio button called located under the “lstFrames” listbox. Add the following event handler: Click="frameRadioButton\_Click".
5. Open **MainPage.xaml.cs** and examine the fields at the top of the class.
6. Add the following code to the **OnNavigatedTo** method (instead of the placeholder). This code initializes the camera, registers to its various events and then sets it as the source for the video brush displayed on the main page:

if(null == camera)

{

    camera = new PhotoCamera();

    camera.Initialized += camera\_Initialized;

    //Event is fired when the button is half pressed

    CameraButtons.ShutterKeyHalfPressed += camera\_ButtonHalfPress;

    //Event is fired when the button is fully pressed

    CameraButtons.ShutterKeyPressed += camera\_ButtonFullPress;

    //Event is fired when the capture sequence is complete and an image is available.

    camera.CaptureImageAvailable += camera\_CaptureImageAvailable;

    camera.CaptureCompleted += camera\_CaptureCompleted;

    //Set the VideoBrush source to the camera

    viewfinderBrush.SetSource(camera);

}

1. Add the following code to the **OnNavigatedFrom** method (instead of the placeholder). This code performs cleanup before leaving the main page:

CameraButtons.ShutterKeyPressed -= camera\_ButtonFullPress;

CameraButtons.ShutterKeyHalfPressed -= camera\_ButtonHalfPress;

pumpEffectedFrames = false;

camera.Dispose();

camera = null;

1. Let us start reviewing the different handlers for various camera events. We will begin with the **camera\_Initialized** handler. This handler simply examines the camera’s available resolutions, picks the first one that is 640 pixels wide and simulates an effect selection by calling **EffectSelected**.
2. **EffectSelected** changes the display according to the selected effect. If no effect is selected (this is implemented by the “echo” effect) then the live video feed is displayed. Otherwise, a frame may be selected (and displayed) and the user may have chosen to apply an effect to the live feed. If an effect was indeed applied, the live feed is replaced with a bitmap that we will update from code using altered live feed information according to the selected effect. This may require us to start the image processing thread and so we call **StartImageProcessing**.
3. **StartImageProcessing** simply initializes the bitmap we use to store the processing result and initializes the thread that pumps data from the camera and processes it. This thread logic is implemented in the **PumpEffectFrames** method.
4. **PumpEffectFrames**  is simply a loop that updates the preview image using a call to **PushProcessedFrame** with some synchronization in place to avoid updating the preview while a picture is being taken.
5. **PushProcessedFrame** and **GetPreviewBuffer** are both used to get the preview buffer from the camera as 32-bits of ARGB data, apply the current effect to them and then place the result in the preview bitmap.
6. Jump to one of the effect classes (the negative effect is fairly simple) to show how an effect manipulates the data from the preview buffer.
7. Moving back to the other camera event handlers we can take a look at **camera\_ButtonHalfPress**,whichcauses the camera to focus, and **camera\_ButtonHalfPress**, which tells the camera to pause all further updates to the preview image and then uses the PhotoCamera instance to take a picture.
8. Once an image is available after the capture, **camera\_CaptureImageAvailable** is invoked, applying the selected effect to the image and saving it to both the image hub and the preview area in the application itself. Later, **camera\_CaptureComplete** is called to restore the preview image updates.

# Summary

In this video, we walked you through an application that utilizes a live camera feed and saw how we can alter this feed in real-time to supply the end user with a unique experience. I hope to see you in our other videos.