Executive Summary
In this application note, I provide a tutorial for creating a Windows application that interfaces with a Kinect for Windows v2 sensor. The Kinect for Windows v2 sensor is a recently developed product with important capabilities, and it is important for consumers to understand how to implement Kinect functionality into their applications. Tutorials from Microsoft and on the web are either incomplete or in video form; I provide a brief, concise documentation. My tutorial will create a Windows application in C# that reads the depth sensing data from the Kinect for Windows v2 sensor and streams an image based on that data to an output window.

Keywords
Windows application, Kinect, depth sensing, C#
Introduction
The Kinect for Windows v2 sensor is a device that can sense depth, capture color images, emit infrared, and input audio [1]. More specifically, the sensor is capable of sensing the depth at a 30 Hz frequency and 512 x 424 resolution and for points further than 0.5 meters from the sensor. The color camera captures 1080p video, also at 30 Hz, and the infrared emitter has similar capabilities. The Kinect for Windows v2 sensor improves on the first version of the device, providing the designated technical spec but also improved body tracking, an expanded field of view, and higher depth fidelity [1]. The sensor used for the Xbox One, but can also be connected to and run on a Windows computer. All of these facts combine to form the notion that the Kinect for Windows v2 sensor (hereafter simply called the Kinect) is a very powerful sensor for use in computer vision and human computer interaction technologies.

Microsoft has developed multiple applications for interfacing with the Kinect. Primarily, it developed the Kinect for Windows SDK, most recently version 2.0, which interfaces with the second version of the Kinect. The SDK provides tools to develop Kinect-enabled applications for Windows and samples of developed applications, including Kinect Fusion, a 3D object-scanning and modeling program [2]. Upon downloading the SDK, one can simply run a sample and use the provided functionality. However, one can also install the samples to view and modify them in a programming environment such as Visual Studio, a Microsoft developer tool. A crude documentation of the SDK is provided on Microsoft’s website.

While it is interesting to interface with developed applications for the Kinect, and also potentially informative, developing applications from scratch allows for much greater creative and learning opportunities. However, the documentation of the SDK does not provide a strong introduction to developing such applications. Also, the samples are very complex and do not yield informative insights easily. Informal guides to developing such applications appear around various Internet pages, but the information is usually spread across multiple pages, documents, or videos. Also, many guides are developed for the first version of the Kinect sensor, as the Kinect v2 is relatively new. A clear, concise introduction to developing Windows applications with Kinect-enabled functionality is needed.

Objective
The objective of this application note is to provide a step-by-step guide to creating Kinect-enabled Windows applications to someone with no experience working with a Kinect but sufficient experience in programming. Potential beneficiaries of this note could be developers who aims to develop Kinect applications or simply programmers who aim to learn more about the capabilities of the Kinect, among others.

The tutorial will focus on acquiring and displaying the depth sensing data. The application will be written in C#, and pieces of it will draw from online tutorials [3,4]. Specifically, the developed application will read the depth sensing information from the Kinect sensor and will output a stream of the processed information (such that it looks like an image) to a window.

Hardware Requirements
1. Kinect for Windows v2 Sensor, power hub, and USB cabling
2. Computer with the following capabilities
a. 64-bit (x64) processor  
b. 4 GB Memory (or more)  
c. 1.7 GHz (or higher)  
d. Built-in USB 3.0 host controller  
e. DX11 capable graphics adapter

**Software Requirements**  
1. Windows 8 or 8.1 (x64)  
2. Visual Studio 2012 or 2013  
3. Kinect for Windows SDK 2.0

**Steps**  
1. Install required software. Visual Studio 2013 can be downloaded [here](#) and the Kinect for Windows SDK 2.0 can be downloaded [here](#).  
2. Open Microsoft Visual Studio.  
3. Click File > New > Project and under the Visual C# tab, choose WPF Application.  
4. Add a reference to your Kinect library. On the right hand side of the application, in the Solution Explorer, right click on References and search for Kinect. Click on and check the box by Microsoft.Kinect and press OK.  
5. Click on the MainWindow.xaml tab. In this tab, in between the two <Grid> statements, we will initialize an image titled camera. Place the following line of code in this area:

   ```xml
   <Image Name="camera"/>
   ```

   Figure 1 provides a screenshot of this code in the application.

   ![Figure 1. Initialize image](http://example.com/image.png)

6. Click on the MainWindow.xaml.cs tab. This tab will be the main workspace for this application and is shown in Figure 2.
7. At the top of the file, enter the following to add Kinect functionality:

```csharp
using Microsoft.Kinect;
```

8. In the **public partial class** `MainWindow : Window`, we will start to write our code. In `public MainWindow()`, below `InitializeComponent();`, write:

```csharp
this.Loaded += OnLoaded;
```

At this point, a new function **void OnLoaded** should be initialized. If not, initialize the function as such:

```csharp
void OnLoaded(object sender, RoutedEventArgs e)
{
}
```
9. Before we fill in this function, we must get instances of the **KinectSensor** and **DepthFrameReader** classes. These classes are used to interact with the Kinect and to stream Depth data. Below the `public MainWindow()` function write:

```
KinectSensor sensor;
MultiSourceFrameReader reader;
```

10. Finally, we will fill **void OnLoaded**. In the brackets, place the following code:

```
this.sensor = KinectSensor.GetDefault();
this.sensor.Open();

this.reader = this.sensor.DepthFrameSource.OpenReader();
this.reader.FrameArrived += OnFrameArrived;
```

This code will get the default sensor, open that sensor, create and open a depth frame reader, and call the method **OnFrameArrived** when a frame arrives (event happens). The current workspace of application up to this point is shown in Figure 3.
11. Similar to when we created `void OnLoaded`, we must create `void OnFrameArrived`. This function will acquire and process the frame every time a frame arrives. We must use the `using` syntax to acquire the frame, so that the frame is handled correctly. If the frame is not `null`, we will pass the frame to a function `ProcessFrame`. `ProcessFrame` will convert the depth frame to a `BitmapSource` image, and we will set `camera` in the output window to this image. Place this method after `void OnLoaded`:

```csharp
void OnFrameArrived(object sender, DepthFrameArrivedEventArgs e)
{
    using (var frame = e.FrameReference.AcquireFrame())
    {
        if (frame != null)
        {
            camera.Source = ProcessFrame(frame);
        }
    }
}
```

A screenshot of this function in the application is shown in Figure 4.
12. Finally, we must create the `ProcessFrame` method. The goal of `ProcessFrame` is to convert the depth frame into an image that displays pixels furthest away as black, pixels closest as white, and pixels in between as shades of grey in between black and white. This method was acquired from a webpage [3].

```csharp
private ImageSource ProcessFrame(DepthFrame frame)
{
    int width = frame.FrameDescription.Width;
    int height = frame.FrameDescription.Height;
    PixelFormat format = PixelFormats.Bgr32;

    ushort minDepth = frame.DepthMinReliableDistance;
    ushort maxDepth = frame.DepthMaxReliableDistance;

    ushort[] pixelData = new ushort[width * height];
    byte[] pixels = new byte[width * height * (format.BitsPerPixel + 7) / 8];

    frame.CopyFrameDataToArray(pixelData);

    int colorIndex = 0;
    for (int depthIndex = 0; depthIndex < pixelData.Length; ++depthIndex)
    {
        ushort depth = pixelData[depthIndex];

        byte intensity = (byte)(depth >= minDepth && depth <= maxDepth ? depth : 0);

        pixels[colorIndex++] = intensity; // Blue
        pixels[colorIndex++] = intensity; // Green
        pixels[colorIndex++] = intensity; // Red

        ++colorIndex;
    }
}
```

![Figure 4. OnFrameArrived](image)

```csharp
void OnFrameArrived(object sender, DepthFrameArrivedEventArgs e)
{
    using (var frame = e.FrameReference.AcquireFrame())
    {
        if (frame != null)
        {
            camera.Source = ProcessFrame(frame);
        }
    }
}
```
int stride = width * format.BitsPerPixel / 8;

return BitmapSource.Create(width, height, 96, 96, format, null, pixels, stride);
}

13. Now, the application should be finished. Build and debug the application, and a streaming depth image should appear, similar to the one in Figure 5. If not, please check your code and that you correctly followed the steps.

![MainWindow](image)

**Figure 5. Output window**

**Conclusion**
This application note provides a step-by-step guide to creating a Windows application in C# that can acquire depth frame data from a Kinect for Windows v2 camera, process the data, and stream the processed data to an output image. It is intended as an introduction to interacting with the Kinect in Microsoft Visual Studio. The Kinect has many more features, including color capture, infrared emission, and face and body recognition, and with this tutorial users should have a strong foundation to start working with these other functionalities.

**References**