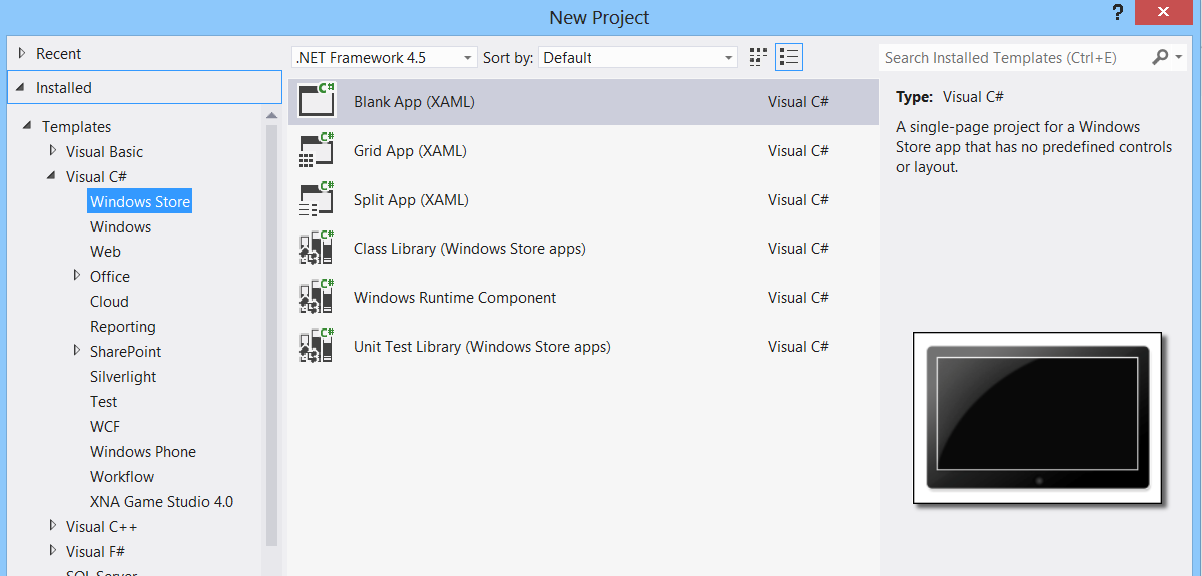
# Connecting to Bluetooth Devices

During Build 2013 there was a great presentation on connecting to Bluetooth devices using the new Windows 8.1 RT API. In the previous WinRT communicating with a Bluetooth device was not available – and even in windows was fairly involved.

In this article I will recreate the session with a few code samples. We will connect to a Sphero device, and send a message to it. You can imagine that in a business application the Bluetooth device could be a printer, scanner, or a card reader instead of a Sphero.

# Project Setup

We will create a very simple C# Windows Store application to communicate with the Sphero. From within Visual Studio 2013, create a new C# Windows Store Application



In the XAML add the following buttons:

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">

<StackPanel Orientation="Vertical">

<Button Click="Button\_Connect">Connect</Button>

<Button Click="Button\_Disconnect">Disconnect</Button>

<Button Click="Button\_Color">Change Color</Button>

</StackPanel>

</Grid>

In the code behind add the following event handlers for the click events for the buttons:

private void Button\_Connect(object sender, RoutedEventArgs e)

{

}

private async void Button\_Color(object sender, RoutedEventArgs e)

{

}

private void Button\_Disconnect(object sender, RoutedEventArgs e)

{

}

Note: I know – where’s the MVVM, the commands, etc. This exercise is about Bluetooth device connectivity.

# Enter the Sphero

Pair the Sphero with your tablet. The device is now ready to be connected to by an application.

In order to allow access to the device the application will need to define it in the application manifest. At the time of this writing, users will need to add the device capability manifest entries in the XML as follows:

<!–- We'll use the Serial Port Profile service on any device. -->

<m2:DeviceCapability Name="bluetooth.rfcomm">

<m2:Device Id="any">

<m2:Function Type="name:serialPort"/>

</m2:Device>

</m2:DeviceCapability>

The capability defines which device we are trying to load, and how we are going to use it.

|  |  |
| --- | --- |
| <m2:DeviceCapability Name="bluetooth.rfcomm"> | This describes to the app that we will be connecting to a Bluetooth device over the RFComm protocol. |
| <m2:Device Id="any"> | This defines that we will connect to any of the Bluetooth devices that offer RFComm. A user can provide a device id to connect to a specific device via “vidpid: xxxx xxxx” |
| <m2:Function Type="name:serialPort"/> | This defines that we will be using a serial port protocol over RFComm |

# Enumeration

In this section we will enumerate the Bluetooth devices that have the relevant settings. By using the following API users will be given a list of available Bluetooth devices:

private async void EnumerateDevices()

{

try

{

var aqs = RfcommDeviceService.GetDeviceSelector(RfcommServiceId.SerialPort);

var serviceInfoCollection = await DeviceInformation.FindAllAsync(aqs);

foreach(var serviceInfo in serviceInfoCollection)

{

devInfo = serviceInfo;

}

}

catch(Exception ex)

{

System.Diagnostics.Debug.WriteLine("Error enum Devices ex=" + ex.ToString());

}

}

Add that API to the Connect click handler as follows:

private void Button\_Connect(object sender, RoutedEventArgs e)

{

EnumerateDevices();

}

When the user clicks on Connect, this code will attempt to find the device.

Using the correct device ID you will request a handle to the device as follows:

private async void StartDevice()

{

try

{

rfcommService = await RfcommDeviceService.FromIdAsync(devInfo.Id);

if (rfcommService != null)

{

//Do Something

}

else

{

System.Diagnostics.Debug.WriteLine("rfcommService==null");

}

}

catch(Exception ex)

{

System.Diagnostics.Debug.WriteLine

("Start Device exception ex=" + ex.ToString());

}

}

To your Connect click handler, add the following:

private void Button\_Connect(object sender, RoutedEventArgs e)

{

EnumerateDevices();

**StartDevice();**

}

…and that’s it – you are now connected to the Bluetooth device.

I asked myself if it could actually be this easy to connect to a device. The answer is – Yes, the WinRT 8.1 API makes it very easy to connect to a device. In just a few lines of code we have connected, and are ready to talk to the device.

If you watched the video presentation at build, you’ll know that everything I’ve written so far follows along with what you see in the video – right down to the part where all you can see is the API calls:

byte[] packet = CreateColorChangePacket( GenerateRandomColor() );

writer.WriteByte(packet);

await writer.StoreAsync();

The presenter did not include the code for the CreateColorChangePacket API in with the presentation. Sphero does provide an SDK which abstracts all the packet building – but I didn’t want to use the SDK, I wanted to really find out how to communicate with the device.

It turns out – that connectivity is about 20% of the equation, the biggest part of the equation is communication. For this, you will require a programming specification for the Sphero. The specification tells you what each API to the Sphero is – along with the byte structure of the API. It will also explain when to send the bytes and when to read the bytes.

# Communicating with the Sphero

In order to communicate with the Sphero we’ll need to take the handle generated in the last section and attach a socket to it – with both a stream writer and a stream reader. This will allow us to send messages to the sphero as well as receive messages from the sphero.

Add the following code to your StartDevices() method:

if (rfcommService != null)

{

socket = new StreamSocket();

await socket.ConnectAsync(rfcommService.ConnectionHostName,

rfcommService.ConnectionServiceName,

SocketProtectionLevel.BluetoothEncryptionAllowNullAuthentication);

writer = new DataWriter(socket.OutputStream);

bConnected = true;

System.Diagnostics.Debug.WriteLine("Device Connected");

}

else

{

System.Diagnostics.Debug.WriteLine("rfcommService==null");

}

This code will setup the reader and the writer. Don’t forget to clean up after the device – you will need to add the following code to the disconnect click handler:

private void Button\_Disconnect(object sender, RoutedEventArgs e)

{

Disconnect();

}

And:

private void Disconnect()

{

bConnected = false;

if(writer!=null)

{

writer.DetachStream();

writer = null;

}

if(socket!=null)

{

socket.Dispose();

socket = null;

}

if(rfcommService!=null)

{

rfcommService = null;

}

}

The disconnect code should be called whenever the device is no longer going to be used. This should be done on application suspending. For example, in the constructor add:

App.Current.Suspending += Current\_Suspending;

And then add the method:

void Current\_Suspending(object sender,

Windows.ApplicationModel.SuspendingEventArgs e)

{

Disconnect();

}

The stream reader will need to be setup on a separate thread so that it can receive messages sent asynchronously from the Sphero. The Sphero may occasionally send unsolicited messages as well as messages in response to requests. In this example, I’ve started a very simple thread that waits for, and reads responses. The code doesn’t care what the response is. More formal code will need to decode the response, and act upon it.

Add the following code to the Main.xaml.cs class:

private async void WaitForData()

{

if (bConnected == false)

return;

var dataReader = new DataReader(socket.InputStream);

dataReader.InputStreamOptions = InputStreamOptions.Partial;

try

{

var payload = await dataReader.LoadAsync(256);

if (payload != 0)

{

var bytes = new byte[payload];

dataReader.ReadBytes(bytes);

System.Diagnostics.Debug.WriteLine("Received Length=" + bytes.Length);

//Piece together a message

DebugBytes(bytes);

}

}

catch(Exception ex)

{

System.Diagnostics.Debug.WriteLine("Exception reading stream");

}

dataReader.DetachStream();

WaitForData();

}

And to the StartDevice() method add:

if (rfcommService != null)

{

socket = new StreamSocket();

await socket.ConnectAsync(rfcommService.ConnectionHostName,

rfcommService.ConnectionServiceName,

SocketProtectionLevel.BluetoothEncryptionAllowNullAuthentication);

writer = new DataWriter(socket.OutputStream);

bConnected = true;

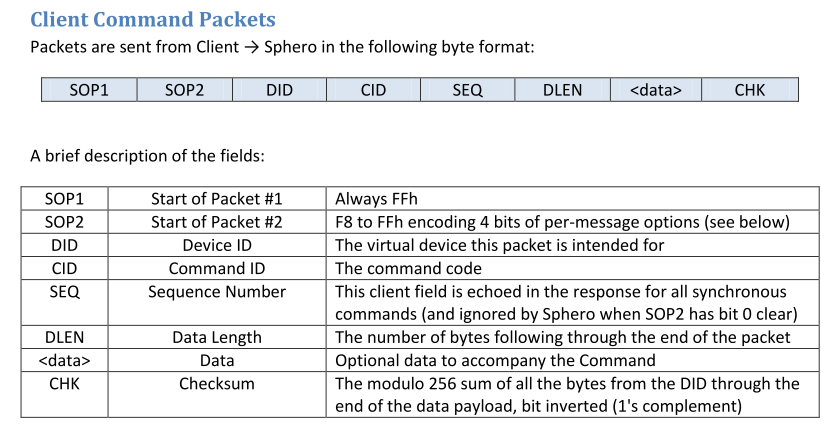
System.Diagnostics.Debug.WriteLine("Device Connected");

**//Start receieve thread**

**WaitForData();**

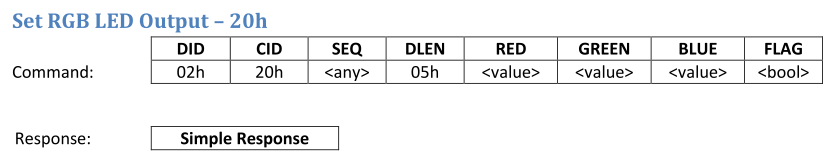
}

We are now ready to send the Sphero a message. From the programming spec of the sphero we know that the packet format is:



Note: You can obtain the full programming spec from Sphero.

I’ll pick the very simple sample message to change the color:



I’ve created a simple function that will build a packet for me as defined in the sphero spec. Add the following code to the application:

In the MainPage.xaml.cs:

Add the private variable:

private SpheroCommands sc = new SpheroCommands();

To the color change click handler add:

private async void Button\_Color(object sender, RoutedEventArgs e)

{

byte[] packet = sc.CreateColorChangePacket(r, g, b);

System.Diagnostics.Debug.WriteLine("Output:");

DebugBytes(packet);

writer.WriteBytes(packet);

await writer.StoreAsync();

if(r==0xff)

{

r = 0;

g = 0xff;

b = 0;

}

else if(g==0xff)

{

r = 0;

g = 0;

b = 0xff;

}

else if(b==0xff)

{

r = 0xff;

g = 0;

b = 0;

}

}

And for debugging:

private void DebugBytes(byte[] bytes)

{

StringBuilder strBuilder = new StringBuilder();

foreach (byte b in bytes)

{

strBuilder.AppendFormat("{0:x2} ", b);

}

System.Diagnostics.Debug.WriteLine(strBuilder.ToString());

return;

}

If you run the application you should be able to click on connect. You’ll see some messages popup from the sphero in the output window. Click on change color and you should be able to see the color change on the sphero.

Basically that’s it – any serial communication over Bluetooth using RFComm is just as simple as this.