Controlling Sphero with Windows Phone 8

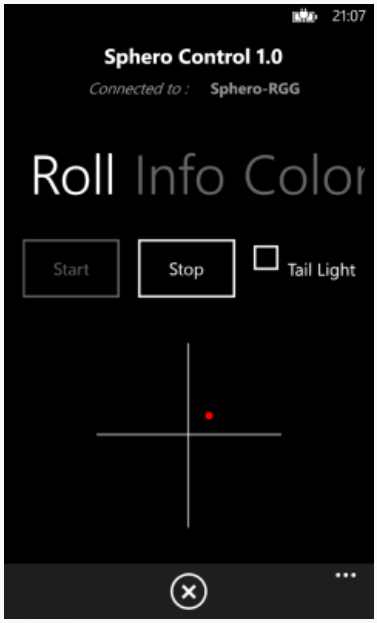
This article explains how to control a Sphero ball using Bluetooth on Windows Phone 8.

Introduction

Sphero is a ball than can be controlled by Bluetooth on Windows Phone 8 (and other mobile OS, using their respective SDKs). The SDK allows you to control the lights, move the ball, manage collision detection, create macros, and perform other tasks.

This code example shows how to control lights and move the ball. There is a fully working Visual Studio solution attached, and screenshots of the demo app are shown below.

Calibration screen



Motion control

About the APIs

You can find the documentation here: https://github.com/orbotix/DeveloperResources/tree/master/docs

Warning: The documented API is not exactly the same as in the SDK. Problems with the GetVersioning command are discussed later.

Command packet

The format of the command packet is shown below.

Client Command Packets

Packets are sent from Client → Sphero in the following byte format:

SOP1	SOP2	DID	CID	SEQ	DLEN	<data>	CHK

A brief description of the fields:

SOP1	Start of Packet #1	Always FFh
SOP2	Start of Packet #2	F8 to FFh encoding 4 bits of per-message options (see below)
DID	Device ID	The virtual device this packet is intended for
CID	Command ID	The command code
SEQ	Sequence Number	This client field is echoed in the response for all synchronous commands (and ignored by Sphero when SOP2 has bit 0 clear)
DLEN	Data Length	The number of bytes following through the end of the packet
<data>	Data	Optional data to accompany the Command
CHK	Checksum	The modulo 256 sum of all the bytes from the DID through the end of the data payload, bit inverted (1's complement)

For example, the command to get Bluetooth information is: 0x00 (DID) 0x11 (CID) (See API p13). I put SEQ = 1 and DLEN is always 1 and I set the CHK byte to 1 too. So the full command packet message will be : **0xFF,0xFF,0x00,0x11,0x01,0x01,0xF2**

An example of a command packet with some data would be one to set the back light. The command to set the backlight is 0x02, 0x21. The data is a byte giving the brightness value from 0 to 255 (0xFF in hex). We set DLEN = 0x02, and 0x01 for CHK. So in order to set the brightness to 255 the command packet would be: **0xFF,0xFF,0X02,0x21,0x01,0x02,0xFF,0xDA**

Response packet

The response format is as shown below:

Sphero Response Packets

Commands are acknowledged from Sphero → Client in a similar format:

SOP1	SOP2	MRSP	SEQ	DLEN	<data>	CHK

SEQ is the same SEQ byte as in the command sent, so you know to which command it is in the response. Often there is no <data> part as it is just an acknowledgement of the command. Such answer is called SIMPLE ANSWER in the documentation.

For the MSRP, check the APIO document appendix for the values (0x00 = OK)

Stream mode

You can set the message to stream which is faster. Note however that in this case responses are not sent and messages can be lost.

To do this mode set the SOP2 to 0xFE. Example with the set brightness command :
0xFF,0xFE,0X02,0x21,0x01,0x02,0xFF,0xDA

Playing with Sphero

First see Bluetooth on Windows Phone 8 documentation on Dev Center.

As we will be using Bluetooth, we need to set the ID_CAP_PROXIMITY, ID_CAP_NETWORKING capabilities in the manifest. Then, just a few lines of code are needed to connect:

```
PeerFinder.AlternateIdentities["Bluetooth:Paired"] = "";

var pairedDevices = await PeerFinder.FindAllPeersAsync();

// Get the device with "sphero" in the name
var sphero = pairedDevices.FirstOrDefault(d =>
d.DisplayName.ToUpper().Contains("SPHERO"));

 if (sphero == null)
{
     MessageBox.Show("No Sphero detected !");
     return;
```

```
}

try
{
     // open a socket
     spheroSocket = new StreamSocket();
     await spheroSocket.ConnectAsync(spheroName, "1");
}
catch (Exception ex)
{
      //...
}

// Get device name
DeviceName = sphero.DisplayName;
```

Note: The Sphero needs to be paired with the phone before we can connect with it !

Now you've got a socket open where you can send the commands

Sending commands

Here is the generic method I use to send the commands

```
public async Task SendCommand(byte[] command)
{
      // Compute checksum
       byte checksum = 0;

       for (int i = 2; i < command.GetLength(0) - 1; i++)
           checksum += command[i];

       // set the complement to 1 of the checksum at the end of the message
       command[command.GetLength(0) - 1] = (byte)~checksum;

       // stream it

       await _spheroSocket.OutputStream.WriteAsync(command.AsBuffer());

       await _spheroSocket.OutputStream.FlushAsync();

       // Show message in output window
       Debug.WriteLine(BitConverter.ToString(command));
}
```

The command byte array is send as parameter. I sum the bytes, from DID to the one before the last (CHK). I invert it, then I put it at the end of the message.

After, I just write it in the `OutputStream` of the socket. I flush it to be sure.

Some basic commands

Except if I really need the response, these examples are in stream mode (SPO2 = 0xFE)

Set Color

You can set the Sphero color with:

Set RGB LED Output – 20h

	DID	CID	SEQ	DLEN	RED	GREEN	BLUE	FLAG
Command:	02h	20h	<any>	05h	<value>	<value>	<value>	<bool>

I just put the R, G and B values in the message

Set back LED

As Sphero is a ball, there is no real front and back. To know what Sphero considers to be "the back", there is a (blue) backlight. You can set the brightness of it using:

Set Back LED Output – 21h

	DID	CID	SEQ	DLEN	BRIGHT
Command:	02h	21h	<any>	02h	<value>

Get Versioning

This was problematic as the API documentation doesn't seem to be correct. After research, it reflects a new version of the API that does not match the SDK.

Here is what I have on my Sphero (There is a byte on 8th position I have no idea what it is for!)

Get Versioning – 02h

	DID	CID
Command:	00h	02h

	DLEN	<data>
Response:	0Bh	see below

The Get Versioning command returns a whole slew of software and hardware information. It's useful if your Client Application requires a minimum version number of some resource within Sphero in order to operate. The data record structure is comprised of fields for each resource that encodes the version number according to the specified format.

Name	Byte index	Description
RECV	0	This record version number, currently set to ~~02h~~ 01h. This will increase when more resources are added.
MDL	1	Model number; currently 02h for Sphero
HW	2	Hardware version code (ranges 1 through 9)
MSA-ver	3	Main Sphero Application version byte
MSA-rev	4	Main Sphero Application revision byte
BL	5	Bootloader version in packed nibble format (i.e. 32h is version 3.2)
BAS	6	orbBasic version in packed nibble format (i.e. 4.4)
MACRO	7	Macro executive version in packed nibble format (4.4)
~~API maj~~	~~8~~	~~API major revision code this firmware implements~~
~~API min~~	~~9~~	~~API minor revision code this firmware implements~~

Here is a `VersioningInfo` class I made:

```
public class VersioningInfo
{
    public string RecordVersion { get; set; }

    public string Model { get; set; }

    public string HardwareVersion { get; set; }

    public string MainAppVersion { get; set; }
```

```
    public string MainAppRevision { get; set; }

    public string BootloaderVersion { get; set; }

    public string OrbBasicVersion { get; set; }

    public string MacroExecutiveVersion { get; set; }

    public string APIVersion { get; set; }

    static public VersioningInfo FromBytes(byte[] array)
    {
        if ((array == null) || (array.GetLength(0) < 9))
            throw new ArgumentException();

        var stringarray = BitConverter.ToString(array).Split('-');

        var obj = new VersioningInfo
                        {
                            RecordVersion = stringarray[0],
                            Model = stringarray[1] == "02" ? "Sphero" : "Unknown",
                            HardwareVersion = stringarray[2],
                            MainAppVersion = stringarray[3],
                            MainAppRevision = stringarray[4],
                            BootloaderVersion = stringarray[5][0] + "." +
stringarray[5][1],
                            OrbBasicVersion = stringarray[6][0] + "." +
stringarray[6][1],
                            MacroExecutiveVersion = stringarray[7][0] + "." +
stringarray[7][1]
                        };

        return obj;
    }
}
```

So, to send the message and get the response :

```
async public Task<VersioningInfo> GetFirmware()
{
     byte[] getFirmwareCommand = { 0xff, 0xff, 0x0, 0x2 ,0x1,0x1,0xFB};

    await _spheroSocket.OutputStream.WriteAsync(getFirmwareCommand.AsBuffer());

    await _spheroSocket.OutputStream.FlushAsync();

    var buffer = new byte[14];

    // read first 5 bytes to have first part of message
    var b = await _spheroSocket.InputStream.ReadAsync(buffer.AsBuffer(), 5,
InputStreamOptions.None);

    var response = b.ToArray();

    Debug.WriteLine("Answer : " + BitConverter.ToString(response));
```

```
    var messageresponse = response[2];

    // read rest of message
    b = await _spheroSocket.InputStream.ReadAsync(buffer.AsBuffer(), response[4],
InputStreamOptions.None);

    if (messageresponse == 0)
    {
        // OK, so create a VersionInfo of it
        return VersioningInfo.FromBytes(b.ToArray());
    }
    else
    {
        throw new Exception("Error code : " + messageresponse);
    }

    return null;
}
```

I send the message (not in stream mode, as I need the answer).

I get the first 5 bytes of the answer. I get the message response code (the third byte) and I get the rest of the message .If it is success, it should be 9 bytes. If not, just one (the CHK). Anyway, I get the remaining length using the 5th byte of the answer (DLEN).

Now I check if the response is successful or not. If not, I thrown an exception. If success, I create a `VersioningInfo`.

Roll

This is probably the main command.

Roll – 30h

	DID	CID	SEQ	DLEN	Speed	Heading	Heading	STATE
Command:	02h	30h	<any>	05h	<val>	<msb>	<lsb>	<val>

This is straightforward - just set a heading and a speed. Heading is 0 to 359. 0 being straight line on positive Y axis.

```
async public Task Move(int heading, int speed)
{
    if (speed > 255)
        speed = 255;

    if (speed < 0)
        speed = 0;

    if (heading < 0)
        heading += 360;

    if (heading > 359)
        heading -= 360;

    var command = { 0xff, 0xfe, 0x2, 0x30, 0x1, 0x5, 0, 0, 0, 0x1, 0 };

    var directionbytes = BitConverter.GetBytes(heading);

    command[6] = (byte)speed;
    command[7] = directionbytes[1];
    command[8] = directionbytes[0];
```

```
    SendCommand(command);
}
```

First I check if speed and heading are in correct range and then convert the heading into bytes. The heading and speed are then added to the message and sent.

Warning: the STATE byte is set to 1. If you want to make Sphero to stop, it is better to set it to 0, in order to give a smooth deceleration (See appendix C in docs).

Sleep

`Sleep()` turns Sphero off.

Sleep – 22h

	DID	CID	SEQ	DLEN	Wakeup	Macro	orbBasic
Command:	00h	22h	<any>	06h	<16-bit val>	<val>	<16-bit val>

```
public async Task Sleep()
{
    var command = new byte[]{ 0xff, 0xfe, 0x0, 0x22, 0x1, 0x6, 0x0, 0x0, 0x0, 0x0, 0x0,
0 };

    SendCommand(command);
}
```

This is a very simple version of the sleep command. There is no timeout and no macro.

Calibrating

As I said earlier, as Sphero is a ball, there is no real front and back. But we still need to calibrate Sphero to set the heading 0 (straight line).

The way I do it is :

- Set the tail light on so I can have a mark
- Rotate the Sphero by calling the `Roll()` command on a heading with a speed of 0 to align it as you need.

the code example calibration screen is shown below:

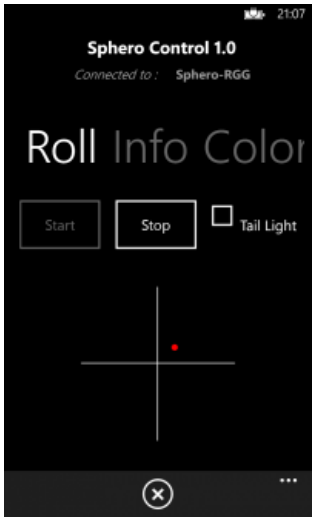
Calibration screen

Keep pressing on the + or – button to move the Sphero by+5 or –5 degrees.

Moving using phone sensors

In my app, you move the Sphero by orienting the mobile. To be more precise, the pitch is for forward/backward and the roll is for left/right

The greater the orientation greater the speed.



You see the red ball in the screenshot. Take a vector from (0,0) to the ball. Its angle gives the heading (0 being the Y axis) and its norm gives the speed.

To get sensor data, I use the Motion class

```
_motion = new Motion();
_motion.TimeBetweenUpdates = TimeSpan.FromMilliseconds(100);
_motion.CurrentValueChanged += _gyroscope_CurrentValueChanged;
_motion.Start();
```

When the orientation changes, I compute the heading and speed:

```
void _gyroscope_CurrentValueChanged(object sender, SensorReadingEventArgs<MotionReading>
e)
{
    //Debug.WriteLine(string.Format("X : {0} Y : {1}", e.SensorReading.Attitude.Pitch,
e.SensorReading.Attitude.Roll));
    var x = e.SensorReading.Attitude.Roll;
    var y = e.SensorReading.Attitude.Pitch;
    var _halfPi = Math.PI/2;

    // Limit to 90 degrees (Pi/2 in radian)
    if (x > _halfPi)
        x = (float)_halfPi;
    if (x < -_halfPi)
        x = (float)-_halfPi;

    if (y > _halfPi)
        y = (float)_halfPi;
    if (y < -_halfPi)
        y = (float)-_halfPi;

    // Move the ball on the screen
    Dispatcher.BeginInvoke(() =>
                                {
                                    ((TranslateTransform)Position.RenderTransform).X =
                                        x * (120 / _halfPi);
```

```
                                    ((TranslateTransform)Position.RenderTransform).Y =
                                        y * (120 / _halfPi);
                                });

    // Get vector angle
    var angle = x != 0.0 ? (int)((Math.Atan(y / x) + _halfPi) * (180 / Math.PI)) : 0;
    // Get vector magnitude
    var speed = Math.Sqrt(x * x + y * y) * 255 / _halfPi;

    // If x < 0, add 180 degrees to have a heading in degrees, clockwise
    if (x < 0)
        angle += 180;

    Debug.WriteLine(string.Format("Angle : {0} Speed : {1}", angle, speed));

    _spheroCommands.Move(angle, (int)(speed));
}
```

I limit orientation to 90 degrees max (pi/2). To get the angle, I use the Atan to get an angle (in radian) from a tangent (y/x). I add 90 degrees (pi/2) as for Sphero, heading 0 (angle 0) is on the positive Y axis, not on the positive X as it is by default for math. Then I multiply by 180/Pi so I have degrees and not radians.

For the speed, the norm is simply square root of (x*x + y*y). I divide by Pi/2 (as Pi/2 is the maximum) in order to have a number between 0 and 1. Then, multiply it by 255 to have a number in the range 0..255.

References

- Build 2013 session about Bluetooth
- Code example app sources: File:SpheroRT.zip
- Original blog and a French translation

Summary

This article is just a quick overview of what Sphero is capable of! There is a lot more functionality available, but even with the simple commands described here it is possible to do a lot.