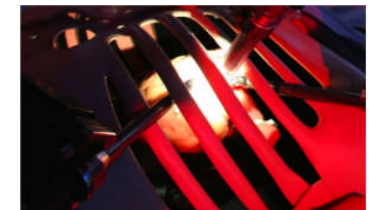
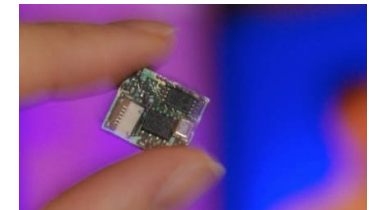
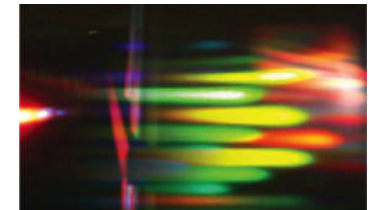




The BSN Hardware and Software Platform: Enabling Easy Development of Body Sensor Network Applications

Joshua Ellul
jellul@imperial.ac.uk



Overview

- Brief introduction to Body Sensor Networks
- BSN Hardware
- BSNOS Installation
- Programming with BSNOS
- Sensor API
- Wireless Sensing API
- Where we're heading



Why BSN?

- 500,000+ places in Residential and Nursing Care homes
 - 400,000 households receive Home Care
 - Costs about GBP 11.1 billion per year, rising by 35% by 2021
 - Home care GBP 150/wk vs. residential or nursing care GBP 500/wk
-
- “The Residential Care and Nursing Home Sector For Older People: An Analysis of Past Trends, Current and Future Demand,”, Department of Health Report, August 2002.
 - “With Respect to Old Age,” Royal Commission on Long Term Care Report, March 1999.



BSN vs WSN?

Challenge	WSN	BSN
Scale	Large as the environment	Size of the human body
Node number	Large number of nodes	Few number of nodes
Node size	Small size preferable, but not a major limitation	Miniaturisation required
Data rates	Application dependent, but commonly low	High
Context Awareness	Typically not essential since node placement is static	Very important, since the body physiology is very sensitive to context change
Power Supply	Minimal	Much less than that of WSN due to size restrictions



The BSN Hardware



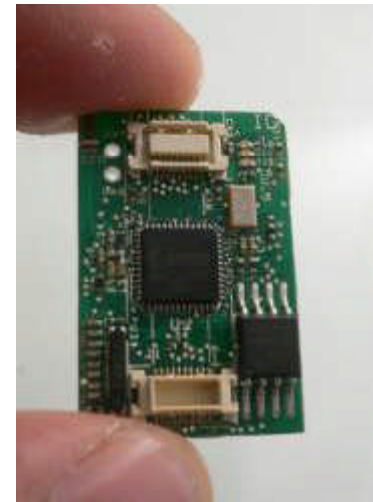
The BSN Hardware (cont.)

- Objective: Platform for enabling the development of wearable wireless sensors for medical and other body related applications.
- Design criteria: Miniaturised, low cost, low power consumption, wireless communication capability, intelligent, extendable, flexible, easy sensor integration.
- Modular stackable hardware, allowing for further extensions.



The BSN Hardware (cont.)

- TI MSP430 16 bit microcontroller
- Chipcon CC2420 low power wireless transceiver
- 4MB Flash memory
- 8 analog channels
- Available Sensors:
Accelerometer, Gyroscope, Magnetometer
Other custom sensors can be integrated



BSNOS

- Many scientists working in medical, sports and other body related fields do not have the low level embedded development skills required to develop body sensor network applications (even many of the computer scientists in the field).
- Even performing simple sensor tests takes time due to most of the work being passed on to a few embedded developers.
- Thus, we have decided to create a framework to enable ease of programming simple sensor network applications.
- Currently, it supports an easy to use API through Java, however any language can be applied.



BSNOS (cont.)

- Furthermore, we will extend the work by providing a GUI block based programming environment for those with no programming experience.



Installation

- (Linux distribution available, Mac port will be available in October)
- Copy the BSNOSIDE directory to your desired location on your pc.
- Plug in the BSN USB board, and wait for the driver to install. If this is not done automatically, then the file is located in the BSNOSIDE\drivers directory.



BSNOS Notes

- Java
 - Run-time compilation - No interpretation overhead
- 16 bit stack width (not 32 bit)
 - Use 'short' variables rather than 'int'
- 'Objects' add a layer of abstraction, which requires more memory and computation. Use static fields and methods as much as possible.
- Language independent
 - Own bytecode

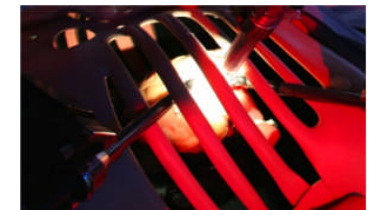
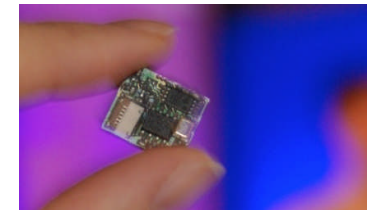
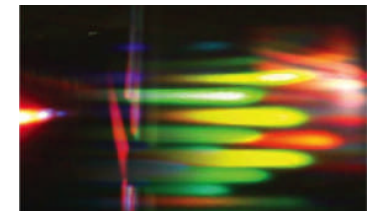




BSNOS Tutorial

Where we're going

Joshua Ellul
jellul@imperial.ac.uk



What is BSNOS?

- An active development project in its beta stages.
- An operating system to facilitate body sensor network application development
- Toolset
- What it is not
 - A networking platform (currently)



Future

- Full Object Oriented support
- Plug-and-play code
- Open Community
 - Library sharing
- Multi-hop support
- Open sensor interface



Get Involved

- New platform
 - A lot of opportunity to contribute
 - Be the first – help your citations
- Java Programming
- Sensor Design
- MAC Protocols
- Routing Protocols
- UI Design
- Applications
- Compression



Get Involved (cont.)

- Context Awareness
- Debugging
- Distributed Algorithms
- Delay Fault Tolerant Algorithms
- Language extensions
 - Python?
 - Basic?
 - Matlab
- GUI Programming
 - LabView
 - Block Programming
-



Keep in touch

- jellul@imperial.ac.uk
- BSN kit's available
- Tutorials and lessons at your University or Organisation
- Help with your BSN applications

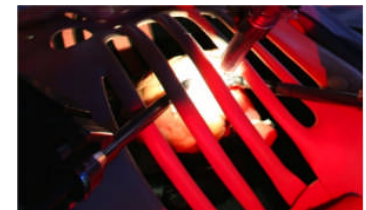
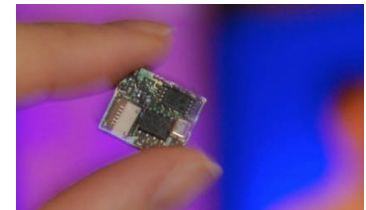
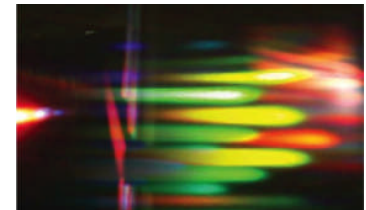




BSNOS Tutorial

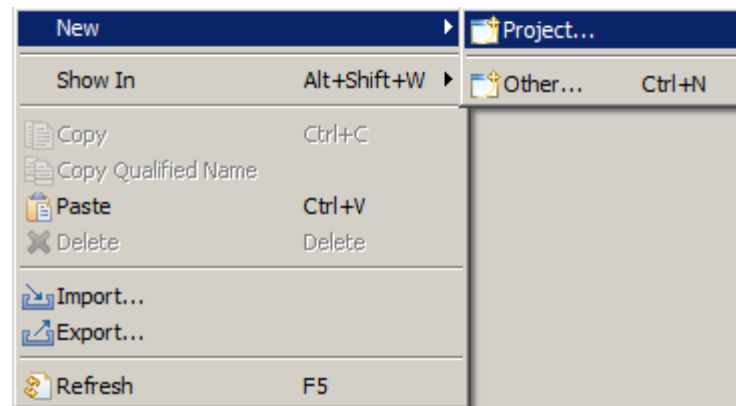
Your First BSN Project

Joshua Ellul
jellul@imperial.ac.uk

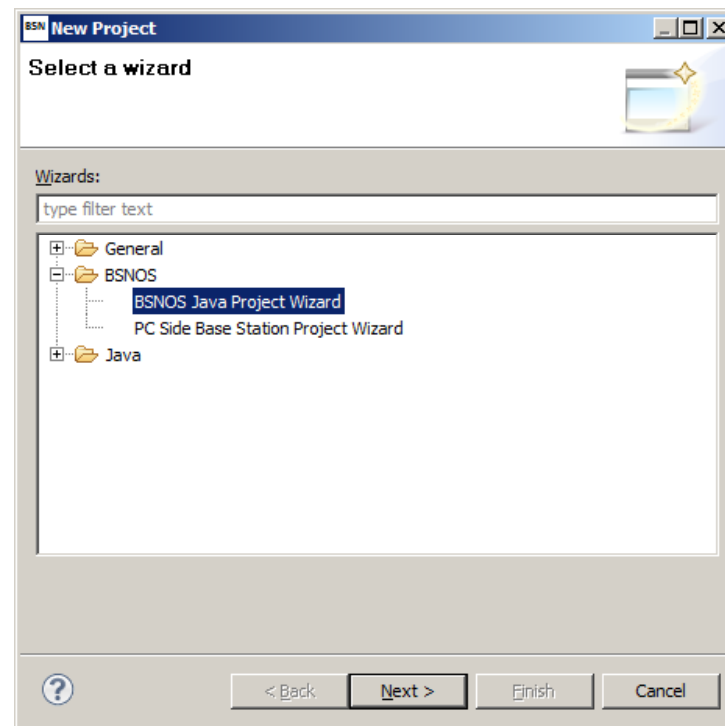


Hello World!

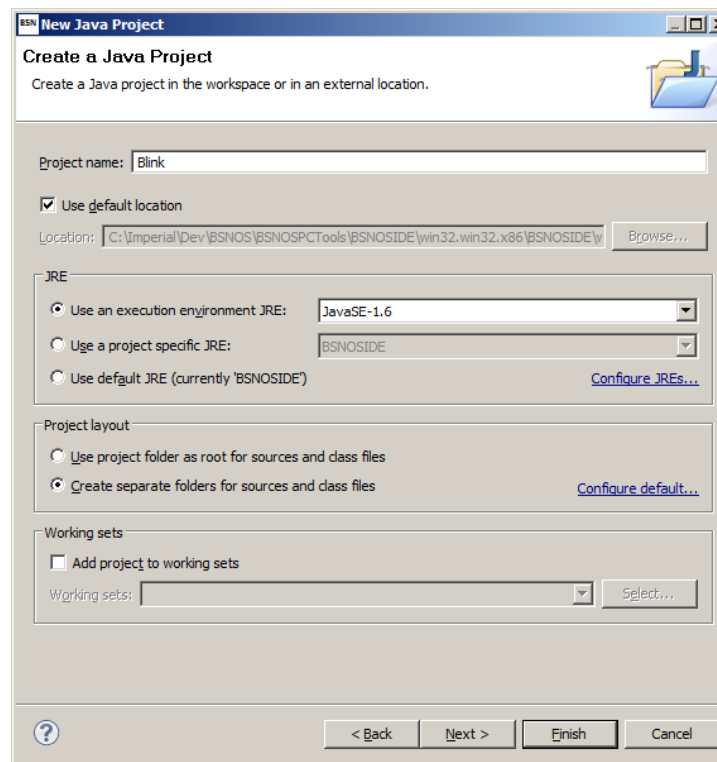
- Let's create a blink application
- In the BSNOS IDE, create a new “BSNOS Java Project” by:
 - Right click in the project explorer (or select the “File” menu)
 - Select the “New” menu item followed by the “Project” menu item



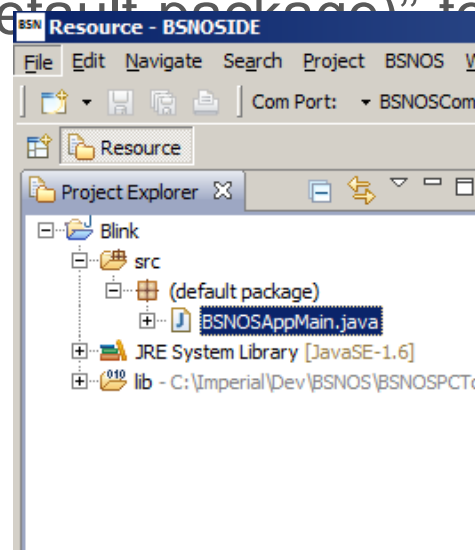
- Expand the “BSNOS” project group and select “BSNOS Java Project Wizard”, and press “Next”.



- Enter the project name “Blink”, and press “Finish”



- The project, “Blink”, will be created and it will be displayed in the “Project Explorer”
 - (if the project explorer is not visible on screen you can show it by selecting the “Window” menu, followed by “Show View” and then “Project Explorer”).
- Expand the “Blink” project to reveal its contents and also expand “src” and “(default package)” to reveal the source code as depicted below:

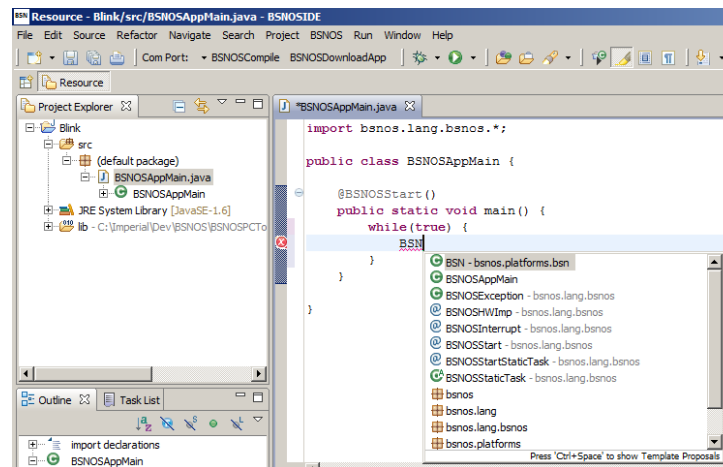


- The file “BSNOSAppMain.java” contains the main execution starting point.
- Open up this file by double-clicking on it. The starting execution point is defined by the annotation “@BSNOSStart()”.
- Let's now add the source code which will toggle an LED and then sleep a while.
- We want the application to perform this action ad infinitum, so we'll wrap the LED toggling and sleeping in a “while loop”.
- So, create a while true loop in the “main()” function as follows:

```
while(true) {  
  
}
```



- The BSN hardware functions are encapsulated inside the “BSN” Java class.
- Type “BSN” on the first line inside the while block, then press “CTRL” and hit the space bar at the same time. This will display the auto-complete feature as shown below:



The screenshot shows an IDE window titled "Resource - Blink/src/BSNOSAppMain.java - BSNOSIDE". The main editor displays the following code:

```
import bsnos.lang.bsnos.*;

public class BSNOSAppMain {

    @BSNOSStart()
    public static void main() {
        while(true) {
            BSN
        }
    }
}
```

An auto-complete dropdown menu is visible below the word "BSN", listing the following options:

- BSN - bsnos.platforms.bsn
- BSNOSAppMain
- BSNOSException - bsnos.lang.bsnos
- BSNOSHWImp - bsnos.lang.bsnos
- BSNOSInterrupt - bsnos.lang.bsnos
- BSNOSStart - bsnos.lang.bsnos
- BSNOSStartStaticTask - bsnos.lang.bsnos
- BSNOSStaticTask - bsnos.lang.bsnos
- bsnos
- bsnos.lang
- bsnos.lang.bsnos
- bsnos.platforms

At the bottom of the dropdown, it says "Press 'Ctrl+Space' to show Template Proposals".



- The first suggestion in the auto-complete popup should be the BSN Java class (bsnos.platforms.bsn).
- Press “.” since we want to access the class' functions to interact with the LEDs. The code to toggle LED number 0 followed by sleeping for 500 milliseconds is displayed below:

```
while(true) {  
    BSN.toggleLed( (byte) 0 );  
    BSN.waitMS( (short) 500 );  
}
```

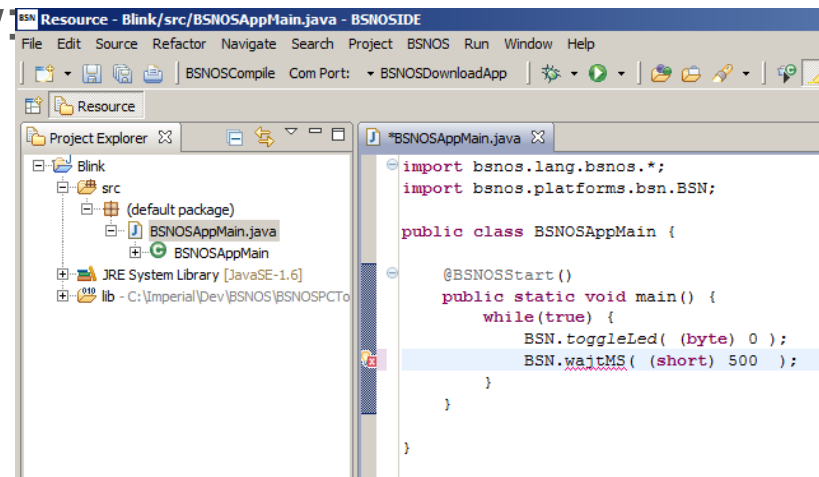


```
while(true) {  
    BSN.toggleLed( (byte) 0 );  
    BSN.waitMS( (short) 500 );  
}
```

- The “toggleLed(byte ledNr)” function is used to toggle the LED (i.e. LED 0).
- The “waitMS(short ms)” function is used to wait a number of milliseconds.
- Since numeric literals are represented by integers by the Java compiler, the LED number must be typecasted to “byte” and similarly the milliseconds specified must be typecasted to “short”.



- If any errors exist in the code they will be underlined in red as show below



```
Resource - Blink/src/BSNOSAppMain.java - BSNOSIDE
File Edit Source Refactor Navigate Search Project BSNOS Run Window Help
BSNOSCompile Com Port: BSNOSDownloadApp
Project Explorer
Blink
  src
    (default package)
      BSNOSAppMain.java
        BSNOSAppMain
JRE System Library [JavaSE-1.6]
lib - C:\Imperial\Dev\BSNOS\BSNOSPCTo

import bsnos.lang.bsnos.*;
import bsnos.platforms.bsn.BSN;

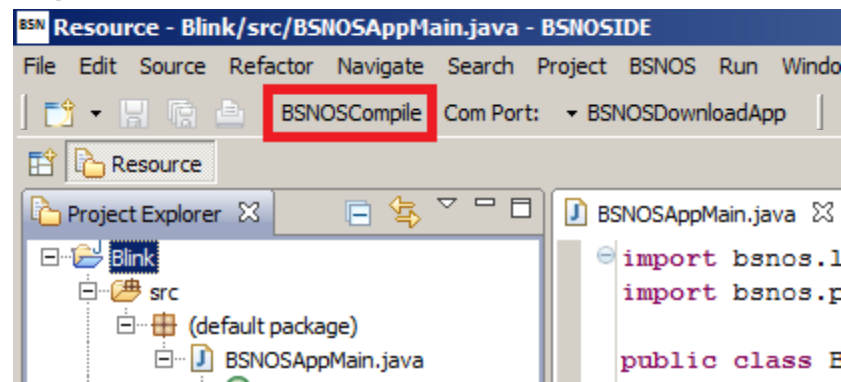
public class BSNOSAppMain {

    @BSNOSStart()
    public static void main() {
        while(true) {
            BSN.toggleLed( (byte) 0 );
            BSN.wajtMS( (short) 500 );
        }
    }
}
```

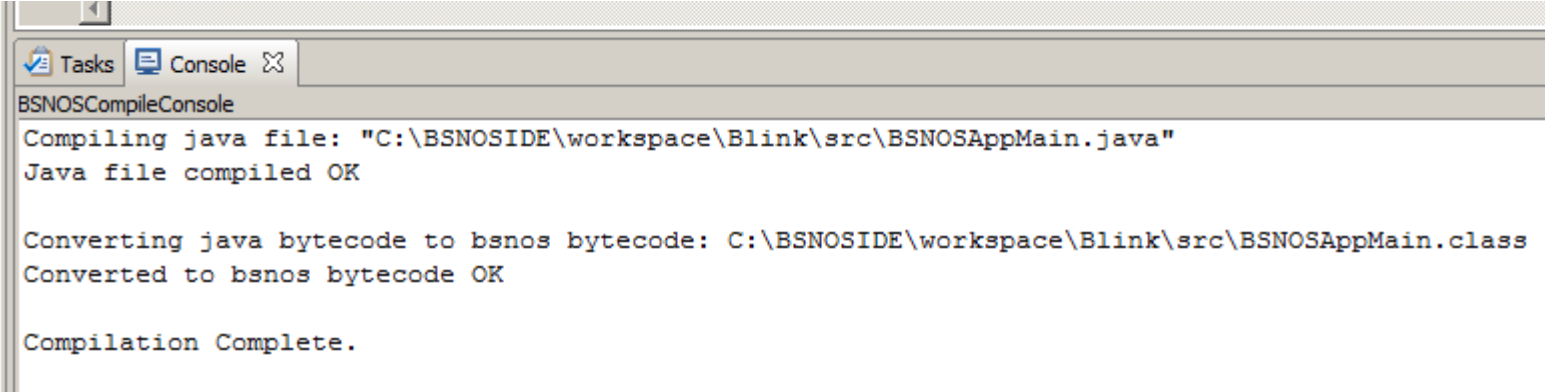
- If any errors exist, the “Problems” window can be displayed to further investigate the error.
- Display this window by selecting the “Window” menu followed by “Show View” and then “Problems”. In the case above, the “waitMS” function was misspelt as “wajtMS”.



- Once, no errors are reported in the source we can compile the code to BSNOS bytecode by selecting the project we want to compile in the “Project Explorer” and then clicking “BSNOSCompile”:



- The “BSNOSCompileConsole” window will display compilation messages. A successful compilation will end with the text "Compilation Complete." as below:



```
BSNOSCompileConsole
Compiling java file: "C:\BSNOSIDE\workspace\Blink\src\BSNOSAppMain.java"
Java file compiled OK

Converting java bytecode to bsnos bytecode: C:\BSNOSIDE\workspace\Blink\src\BSNOSAppMain.class
Converted to bsnos bytecode OK

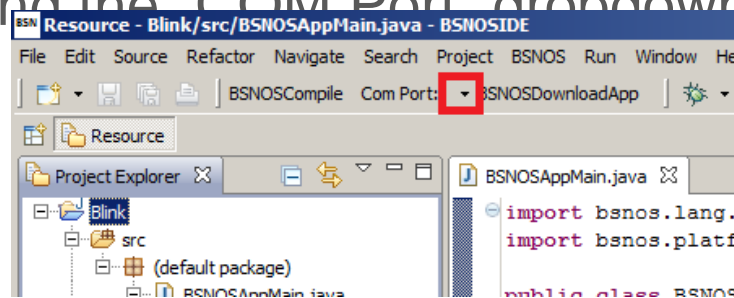
Compilation Complete.
```



- Now, that the code is successfully compiling to BSNOS bytecode, we can program a BSN node.
- Ensure that the BSN USB board is connected and that a BSN main board is on the USB board.
- If you are plugging in the USB board for the first time you may be required to install the drivers for the board (auto-installation of the drivers may work as well).



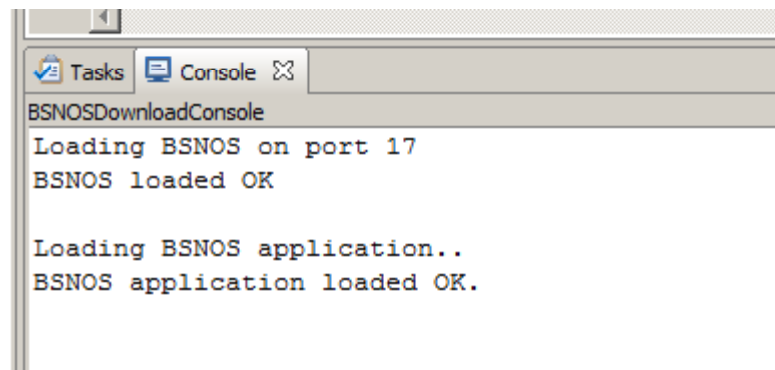
- To download the application to the BSN node first click the project that you would like to download, in this case the “Blink” project.
- Then select the COM port which the USB board is connected to, by clicking the “COM Port” dropdown arrow shown below:



- Then press the “BSNOSDownloadApp” button which is next to the dropdown arrow.



- The “BSNOSDownloadConsole” will display download messages. First BSNOS will be downloaded to the BSN node, followed by the application you have written.
- A successful download will look similar to below:



```
BSNOSDownloadConsole
Loading BSNOS on port 17
BSNOS loaded OK

Loading BSNOS application..
BSNOS application loaded OK.
```

- If the download was successful you should now see one of the LEDs on the main BSN board toggling!



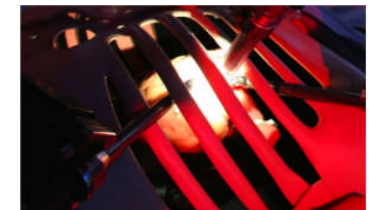
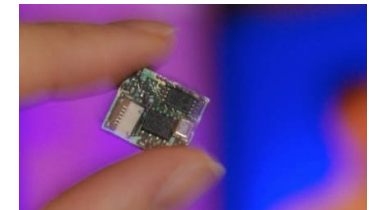
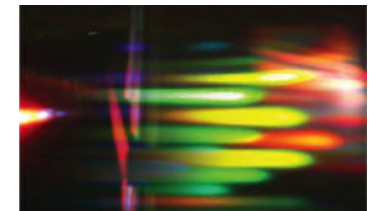


Imperial College
London

BSNOS Tutorial

Sample And Send

Joshua Ellul
jellul@imperial.ac.uk



The Hamlyn Centre
The Institute of Global Health Innovation

The Sample and Send Application

- Sample and Send applications are often sufficient for many Body Sensor Network deployments.
- Let's create a sample and send application that samples the accelerometer and broadcasts the sensed values via the wireless module.
- Start by creating a new “BSNOS Java Project” and name it “SampleAndSend”



Sampling Sensors

- To sample a sensor a single call is required to be made. The following table lists the different sensor sample functions:

Sensor	Function
Accelerometer	<code>BSN.performAccelSample(_)</code>
Gyroscope	<code>BSN.performGyroSample(_)</code>
Magnetometer / Compass	<code>BSN.performMagSample(_)</code>



- After sampling a sensor, the respective sensor's sensed values can be retrieved using

Sensor Value	Function
Accelerometer X	<code>short BSN.getAccelXRaw()</code>
Accelerometer Y	<code>short BSN.getAccelYRaw()</code>
Accelerometer Z	<code>short BSN.getAccelZRaw()</code>
Gyroscope X	<code>short BSN.getGyroXRaw()</code>
Gyroscope Y	<code>short BSN.getGyroYRaw()</code>
Gyroscope Z	<code>short BSN.getGyroZRaw()</code>
Magnetometer X	<code>short BSN.getMagXRaw()</code>
Magnetometer Y	<code>short BSN.getMagYRaw()</code>
Magnetometer Z	<code>short BSN.getMagZRaw()</code>



Building Radio Messages

- The BSNOS API exposes a number of functions to easily build up radio messages.
- Individual values of type byte, short, int and float can be appended to a radio message to be sent.
- The following list is used to append values to a radio message:

Value to append	Function
byte	<code>BSN.appendByteToRadio(byte b)</code>
short	<code>BSN.appendShortToRadio(short s)</code>
int	<code>BSN.appendIntToRadio(int i)</code>
float	<code>BSN.appendFloatToRadio(float f)</code>



Sending Radio Messages

- Once a message has been built and is ready to send, a single call to the following function is used to send the radio message:

```
BSN.sendRadioMsg( short destination_addr )
```

- To send broadcast messages the following constant is used:

```
BSN.BROADCAST_ADDR
```



The Sample and Send Application

- To create the sample and send application, the first step is to sample the sensor. Let's sample the accelerometer by making a call to: `BSN.performAccelSample() ;`

```
@BSNOSStart()  
public static void main() {  
    BSN.performAccelSample( ) ;  
}
```



- The next step is to get the sensor readings sensed and append them to a radio message. So, we will use `BSN.getAccelXRaw` to retrieve the accelerometer's X axis reading, and then append it to a radio message by passing the returned value into `BSN.appendShortToRadio`

```

@BSNOSStart()
public static void main() {
    BSN.performAccelSample();

    BSN.appendShortToRadio(
        BSN.getAccelXRaw()
    );
}

```



- The same is required for the Y and Z axis:

```
@BSNOSStart()  
public static void main() {  
    BSN.performAccelSample();  
    BSN.appendShortToRadio( BSN.getAccelXRaw() );  
    BSN.appendShortToRadio( BSN.getAccelYRaw() );  
    BSN.appendShortToRadio( BSN.getAccelZRaw() );  
}
```



- Now, let's send the radio message as a broadcast message:

```
@BSNOSStart()  
public static void main() {  
    BSN.performAccelSample();  
    BSN.appendShortToRadio( BSN.getAccelXRaw() );  
    BSN.appendShortToRadio( BSN.getAccelYRaw() );  
    BSN.appendShortToRadio( BSN.getAccelZRaw() );  
    BSN.sendRadioMsg( BSN.BROADCAST_ADDR );  
}
```

- The above code samples the sensor and sends the 3 axes' values over the wireless module. Now, to just repeat this



- Now, to wrap the code in a while-true loop so that it is repeated, along with a pause in-between the cycle and a toggling of a LED to indicate that the application is running.

```
@BSNOSStart()  
public static void main() {  
    while ( true ) {  
        BSN.performAccelSample();  
        BSN.appendShortToRadio( BSN.getAccelXRaw() );  
        BSN.appendShortToRadio( BSN.getAccelYRaw() );  
        BSN.appendShortToRadio( BSN.getAccelZRaw() );  
        BSN.sendRadioMsg( BSN.BROADCAST_ADDR );  
  
        BSN.waitMS( (short) 500 );  
  
        BSN.toggleLed( (byte) 0 );  
    }  
}
```



- Set the channel id due to collisions

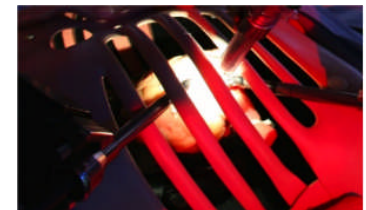
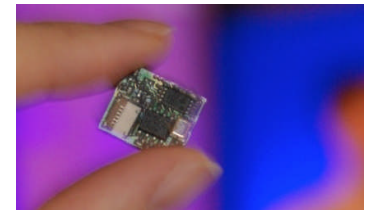
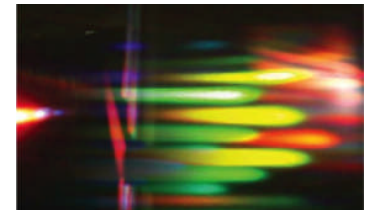




Imperial College
London

BSNOS Tutorial Base Station

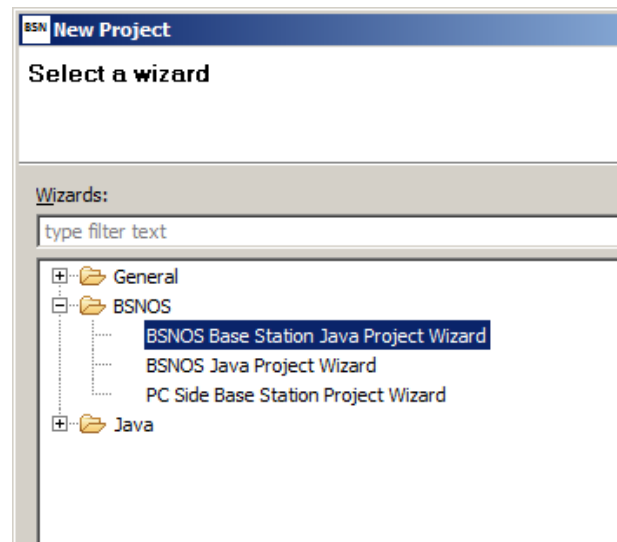
Joshua Ellul
jellul@imperial.ac.uk



The Hamlyn Centre
The Institute of Global Health Innovation

Base Station

- Base stations are mainly used for collecting data from sensor networks and forwarding the received data to the PC
- Let's create a simple base station which forwards messages received to the PC
- Create a new “**BSNOS Base Station Java Project**”



- The default code is enough to have an operating base station.
- The default code performs the following:
 - Listens for incoming messages
 - When a message is received it forwards it over the serial connection, and then toggles an LED.
- The serial protocol used to communicate from base stations to the PC is defined by:
- <STX> <LENGTH> <PAYLOAD> <ETX>



- Now, compile the Base Station project
- Then, if it is successfully compiled, download it to the Base Station node.
- Once downloaded, plug in a node programmed with the “Sample and Send” application and you should see the base station node’s LED blinking. This means that it is receiving data (and attempting to forward it over the serial connection).



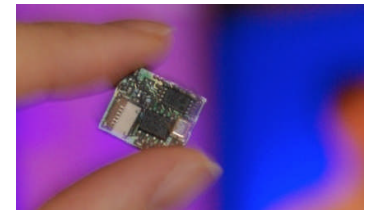
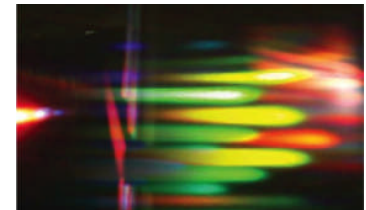


Imperial College
London

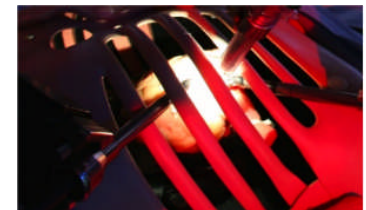


BSNOS Tutorial

PC Side Application



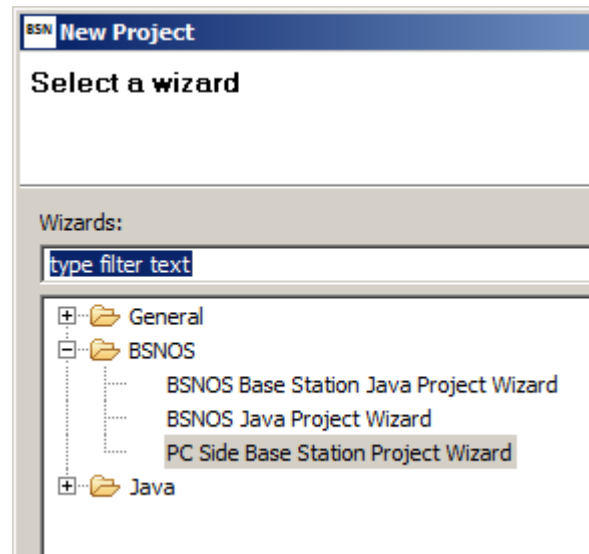
Joshua Ellul
jellul@imperial.ac.uk



The Hamlyn Centre
The Institute of Global Health Innovation

PC Side Application

- Data retrieved in a sensor network is usually meant to be forwarded to a PC for analysis.
- At this stage, we should have a sample and send node, along with a base station node receiving the sampled data.
- To create PC Side application, create a new “**PC Side Base Station Project**”.



- Expand the project in the “Project Explorer”, and then further expand “src” and “(default package)”
- Then open the Java file.
- The “main” function creates an instance of the PC Side thread, which listens to a serial port for incoming data.
- Upon receiving a valid serial message, the “serialPacketReceived” function will be called.
- The “msg” parameter is the data packet received



- Once the data is received, we can now do anything with it like save it to a file, display its contents in a console, or a graph.
- The following is code to dump the received message in HEX to the console:

```
public void serialPacketReceived(byte[] msg) {  
    String s = "";  
    for(byte b : msg) {  
        int bi = b & 0xFF;  
        if (bi < 16) {  
            s += "0";  
        }  
        s += Integer.toHexString(bi);  
        s += " ";  
    }  
    System.out.println(s);  
}
```



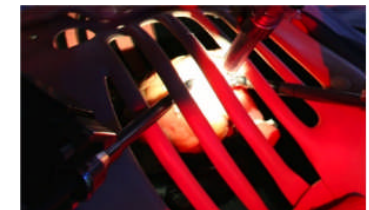
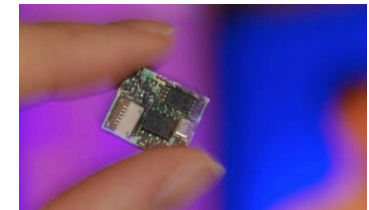
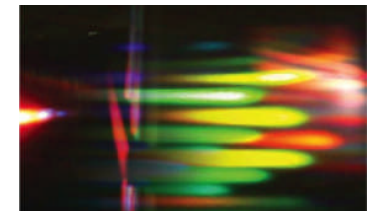


Imperial College
London

BSNOS Tutorial

Making Sense of Messages

Joshua Ellul
jellul@imperial.ac.uk



The Hamlyn Centre
The Institute of Global Health Innovation

Understanding Messages

- The “Sample and Send” application (from tutorial 2) samples the accelerometer and sends the X, Y and Z coordinate over the wireless medium.
- Radio messages also contain other information:
 - Message Length
 - Source Address
 - Destination Address
 - RSSI (Received Signal Strength Indicator)
 - CRC (Cyclic Redundancy Check)



- We will now work on parsing the radio message received.
- Create a new “PC Side Base Station Project”, and open the java file in the “src/(default package)” directory
- The “serialPacketReceived” function is called when a message is received, and the “msg” parameter contains the raw data received.
- To parse the data all we need to do is define the structure of the message.



- Create a new Java class, named “AccelerometerMessage”
- The radio message structure for the Sample and Send application is as follows:

<Dest_Addr> <Source Addr> <Length> <X> <Y> <Z> <RSSI>
<CRC>

- Create “int” fields for each of the message fields as follows:

```
public int destination;  
public int source;  
public int length;  
public int x;  
public int y;  
public int z;  
public int rssi;  
public int crc;
```



- We need to mark each field with the actual datatype for each field. Change your code to mark each field as follows:

```
@BSNOSUnsignedShort  
public int destination;  
@BSNOSUnsignedShort  
public int source;  
@BSNOSUnsignedByte  
public int length;  
@BSNOSUnsignedShort  
public int x;  
@BSNOSUnsignedShort  
public int y;  
@BSNOSUnsignedShort  
public int z;  
@BSNOSUnsignedByte  
public int rssi;  
@BSNOSUnsignedByte  
public int crc;
```



- The “AccelerometerMessage” class provides the message structure. Note: the order of fields are important so they should not be altered.
- To change the raw bytes in the “serialPacketReceived” function to the message structure , and output the coordinates to the console use the following code:

```
AccelerometerMessage m =  
    (AccelerometerMessage) ProtocolHelper.getObjectFromMessage(msg,  
        AccelerometerMessage.class);  
System.out.println("X:\t" + m.x + "\tY:\t" + m.y + "\tZ:\t" + m.z);
```



- Check if the CRC is valid

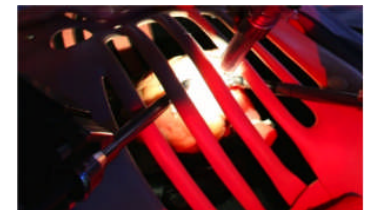
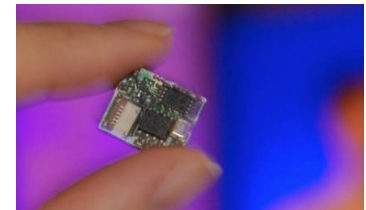
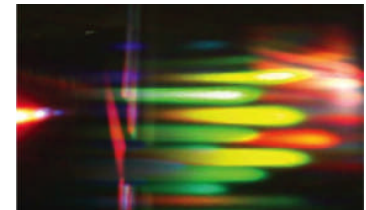




Imperial College
London



Graphing the data



Joshua Ellul
jellul@imperial.ac.uk



The Hamlyn Centre
The Institute of Global Health Innovation

Look at the data

- Looking at the readings can help you to design algorithms
- A graphical UI frame is included in the toolset
 - `MultiSeriesJChart2DFrame`
- Create an instance of the frame, and run it:
 - `graphFrame = new MultiSeriesJChart2DFrame(500);`
 - `graphFrame.start();`
- Add data to the graph:
 - `graphFrame.addReading(0, data);`



Import the charting library

- Import jchart2d
- Build Path > Configure Build Path > Libraries > Add External Jars
 - jchart2d-3.2.0.jar

