

Rochester Institute of Technology  
Computer Science Department

**Investigation Report - Draft#1**

on

**Sun SPOTS  
(Independent Study)**

**Submitted By:**

**Karthik Nathan**

**Karthik Rajagopal**

**Maxat Maketov**

**Mesut Arik**

# Table of Contents

1. Introduction
2. Hardware Characteristics
  - 2.1 Standard Components On a Sun SPOT
    - 2.1.1 Battery
    - 2.1.2 Main board
    - 2.1.3 SPOT Daughter Board
    - 2.1.4 eDemo Board
  - 2.2 Gadgets
3. Programming SPOTs
  - 3.1 Remote Deployment
  - 3.2 Remote Debugging
4. Sun SPOT API
5. Communication Characteristics

## **1. Introduction**

Sun SPOT (Small Programmable Object Technology) is an experimental platform to inspire developers to build wireless sensor networks applications using Sun Technologies. Sun SPOTs are the one of the most powerful wireless sensor network devices in the industry today. And it runs on other well-known and successful Sun Technologies, that is Java, which has been used on over 6 billion devices throughout the world.

A Sun SPOT Device is a small, wireless, battery powered sensor network node, which also contains several other components to sense the environmental and other factors that surrounds it.

## **2. Hardware Characteristics**

This new technology is targeted for developers who are interested in development of wireless sensor networks (WSNs), robotics and other tiny devices running java applications. The SPOTs are powered by low-power battery and run Java 2 Platform, Micro Edition (Java ME). Also, for the SPOTs, Sun Microsystems developers had applied a “sandwich” technology. All the SPOTS are built from almost the same basic elements: Battery, Main board, Daughter board (includes eDemo Board) and plastics keeping parts together. The main difference of various SPOTs is only in the combination of these parts.

### **2.1 Standard Components On a Sun SPOT**

#### **2.1.1 Battery**

The battery is a 3.7 v 720 maH rechargeable lithium-ion prismatic cell. The battery has internal protection circuit to guard against over discharge, under voltage and overcharge conditions. The battery can be charged from either the USB type mini-B device connector or from an external source with a 5V  $\pm$ 10% supply. Typical shelf life losses at room temperature are about 2% of the batteries capacity per month and the

rate can increase with the rise in temperature. Unfortunately, the battery cannot be changed for another type of battery. According to the SPOTS documentation, the charging and power management systems are tuned for this given battery [1]. Eventually, the life of battery is sufficient for multiple purposes. The time during which it can operate in deep sleep is 909 days and by maximum consumption of energy by all 8 LEDs it can run up to full 3 hours of work. [3]

### **2.1.2 Main Board**

The eSPOT main board contains the: main processor, memory, power management circuit, 802.15.4 radio transceiver and antenna, battery connector and daughterboard connector. [1]

### **2.1.3 SPOT Daughter Board**

According to the theory of operation, The SPOT daughter board should satisfy at least three conditions:

1. Connect to the eSPOT main board with a Hirose DF17-30 connector
2. Act as an SPI slave in communication with the eSPOT main board
3. Contain SPI flash memory for storing configuration information

One of the examples of SPOTs daughter boards is the eDemo Board

### **2.1.4 eDemo Board**

The eDEMO board is provided as an example of daughter boards that are compatible with the eSPOT main board. The eDemo board has on it a 3-axis accelerometer, an ambient light sensor, eight tricolor LEDs, two push buttons, six analog input pads, four high current high voltage output pads, and five general I/O pads. [1]

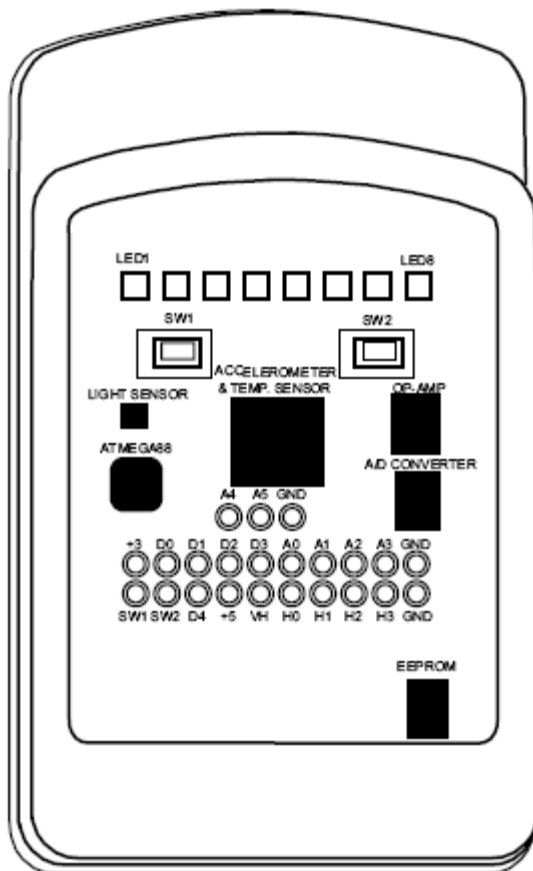
For more information please see in the Sun SPOT theory of operation [1] and Sun SPOT Owner's Manual.

## 2.2 Gadgets

There are a number of gadgets that were implemented with using Sun SPOTS, and most of them are related to robotics. The range of interaction of Sun SPOTs with other hardware could not be measured and is limited to the imagination of the developer and

hardware specifications. Already working results, as the example of servo controller given in the documentation, are available to on various developers' blogs and Sun SPOTs forums [4].

Most of the gadgets are implemented using the eDemo board by using the pins available on the board. Here is the image of Sun SPOT with the eDemo board on the top. On it one can see 8 LEDs (LED1 – LED8), 2 pushbuttons (SW1 and SW2), Light Sensor, Accelerometer and Temperature Sensor, 20 pins with different labels (will be discussed later), Analog to Digital converter (A/D CONVERTER), Operational Amplifier (OP.AMP) and Electrically Erasable Programmable Read-Only Memory (EEPROM)



Each LED can take any color of RGB color space. This ability allows developers to visualize data and various processes in many different ways performing assigning a unique color or sequence of turned on LEDs.

The pushbuttons allow developers get the user input and perform its processing in their applications.

Built in sensors give an ability to gather physical information about the current state of the surrounding environment by measuring the temperature, light and the acceleration in 3-axis.

Finally the main feature of the Sun SPOT eDemo board is the set of 20 pins for various input/output purposes. According to the documentation the set of pins consists of general purpose input/output pins (GPIO), high

current, analog input pins, ground, pins that can be set either to input or output, and others. The following picture from the documentations describe the pins:

<b>V<sub>CC</sub> +3VDC Output 100ma Maximum</b>	<b>SW1</b>	<b>1</b>	<b>2</b>	<b>V<sub>CC</sub></b>
<b>V<sub>+5V</sub> +5VDC Output 100ma</b>	<b>SW2</b>	<b>3</b>	<b>4</b>	<b>D0</b>
<b>V<sub>H</sub> +4.5V to 18VDC Input</b>	<b>D4</b>	<b>5</b>	<b>6</b>	<b>D1</b>
<b>A0-3 Analog Input 10 bit 0V to 3.0VDC</b>	<b>V<sub>+5V</sub></b>	<b>7</b>	<b>8</b>	<b>D2</b>
<b>D0-4 GPIO</b>	<b>V<sub>H</sub></b>	<b>9</b>	<b>10</b>	<b>D3</b>
<b>H0-3 High Current Output 125ma 0V to V<sub>H</sub></b>	<b>H0</b>	<b>11</b>	<b>12</b>	<b>A0</b>
	<b>H1</b>	<b>13</b>	<b>14</b>	<b>A1</b>
	<b>H2</b>	<b>15</b>	<b>16</b>	<b>A2</b>
	<b>H3</b>	<b>17</b>	<b>18</b>	<b>A3</b>
	<b>GND</b>	<b>19</b>	<b>20</b>	<b>GND</b>

**V<sub>CC</sub>** supporting +3V Output with 100 ma maximum comes from the eSPOT.

**V<sub>+5</sub>** is generated on the eDEMO board through the amplifier.

The general purpose digital I/O (D0-4 GPIO) may be configured as either inputs or outputs.

D0 and D1 may also be used as UART data lines RX and TX. D2 and D3 may also be used as I2C-DATA and I2C-CLOCK respectively. Certain restrictions apply.

The high current outputs (H0-H3) are totem-poll capable of sinking and sourcing 125ma. The high voltage output is set by V<sub>H</sub> which is externally connected to P1 pin 9. V<sub>H</sub> (see on the image) is between +4.5V to +18VDC and must be connected to a positive supply for H0-H3 to function.

The eDemo board is capable of inputting of analog signals. For this purposes eDemo board is supplied with A0-A3 pins. The analog to digital converter (ADC) is used to convert analog inputs from the pins A0, A1, A2, and A3. The same ADC is also used to encode analog inputs from accelerometer and light sensor to digital output to the main eDemo processor.

Finally, the GND pins support ground for the connections. With using of this set of pins, developers had developed a plurality of interactions of various hardware on Sun SPOTs. The applications vary from remote controlled cars, pumpkins and other devices to complicated interactions such as recent attempts to connect speech synthesizer.

### 3. Programming SPOTs

Squawk Virtual Machine running on the Sun Spots, is a Java virtual machine that directly runs on top of the Sun SPOTS without any support from any operating system. Sun SPOTS can be programmed using various IDEs such as Netbeans and Eclipse.

Applications made on host machine can be deployed to the sun spots over the air( OTA ) with the help of Base station or any Spot running in Base station mode.



Before deploying any applications to the sun spots remotely, we must make sure that, the remote sun spot is executing OTA( over the air ) command server.

The command server is a background thread which runs on the squawk virtual machine and listens for over the air commands. This thread is started automatically each time the SPOT starts. To enable OTA on any sun spot, issue this command via a USB connection.

*ant enableota*

After an spot has been enabled to run OTA command server, this information is stored in its flash memory, so each time when the spot restarts, it reads this information from its flash memory and decides weather to run OTA or not.

OTA command server can also be disabled using the command,

*ant disableota*

To deploy applications to remote sun spot, basestation can be used or any sun spot can be made to act as a basestation.

To enable a sun spot to run in base station mode, issue the following command to the sun spot via USB

*ant selectbasestation*

Press the control button on the sun spot, and now it will act as a basestation.

### **3.1 Remote Deployment**

After the sunspot is configured to run OTA and command server and appropriate base station has been selected, the following steps are needed to deploy an application from host to remote sun spot over the air with the help of base station connected to the host.

To deploy, issue the following command

```
ant -Dremoteld = <IEEE Address> deploy
```

Where IEEE Address is a 64 bit address of a sun spot whose first 8 bits are common, which is 0014.4F01.xxxx.xxxx

The last eight bits can be seen printed on back of the sun spots.

To run the application, issue the following command

```
ant -Dremoteld = <IEEE Address> run
```

### **3.1 Remote Debugging**

Applications running on the sunspots can be debugged remotely just like an application running on the host. To do so, squawk high level debugger uses Java Debug Wire Protocol(JDWP) which is compatible with IDEs such as Netbeans and Eclipse.

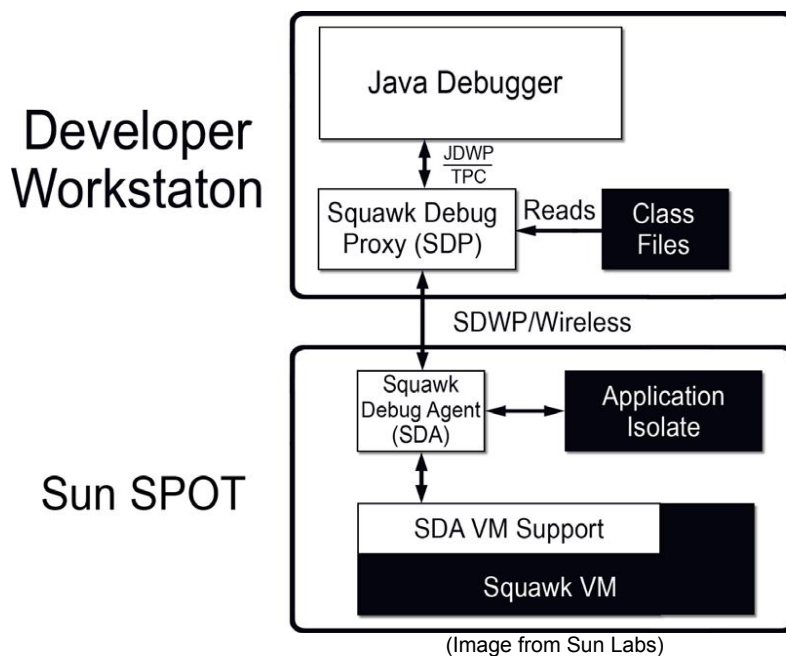
In the current version of the sun spots ( orange release ), it is not possible to debug the application via USB, but can be debugged using a base station connected to the host machine and communicating remotely with the sun spots.

Due to memory constraints on the sun spots, complete JDWP protocol is not implemented on the sun spots, rather the architecture of the sun spots is altered to compensate the shortcomings of not using a complete JDWP protocol.

Architecture of Squawk High level debugger :

- **Squawk Debug Agent ( SDA )** : This agent runs on the sun spot being debugged. This agent controls the execution of the applications running on the sun spots.
- **Squawk Debug Proxy ( SDP )** : This is a proxy program that runs on the host machine. It communicates over the air via base station to the remote spot's SDA.

This kind of architecture allows JPDA( Java Platform Debugger Architecture ) to be located directly on the host machine rather than on the sun spots, thus saving memory and processing requirements of the spots.



To start debugging:

1. Deploy your applications over the air or via USB to the sun spots.
2. Make sure that, the spot is running OTA command server.
3. Attach the base station to the host machine via USB
4. Use the following command to start debug process

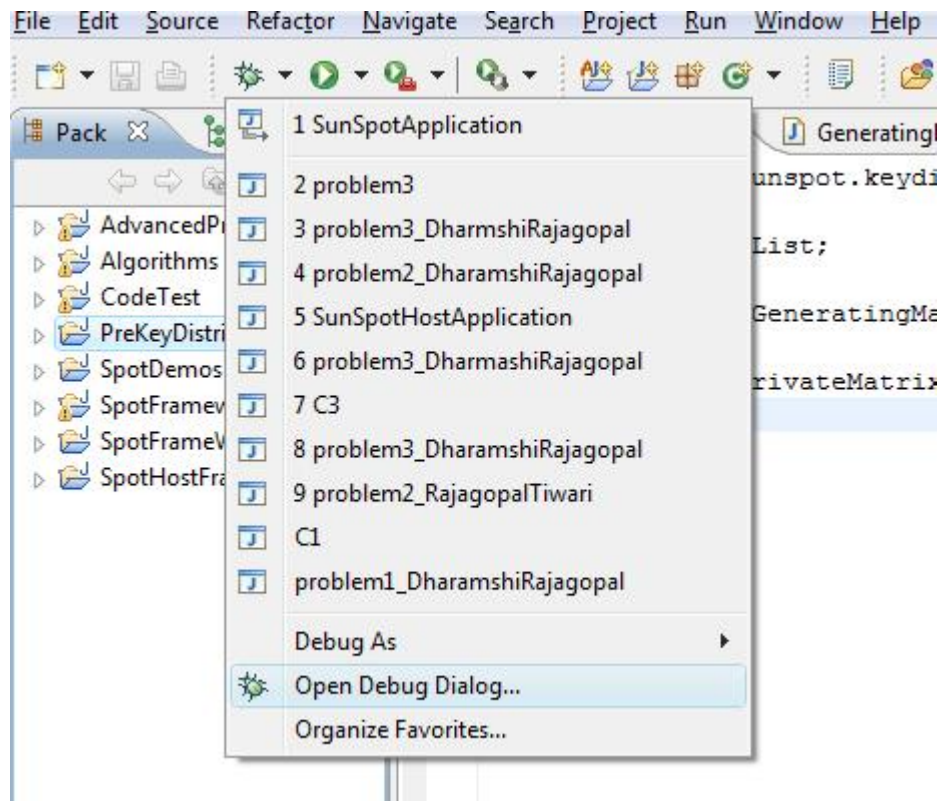
*ant debug -Dremoteld = xxxx -Dbasestation.addr = yyyy*

Where xxxx is the IEEE address of the remote sun spot and yyyy is the MAC address of the base station

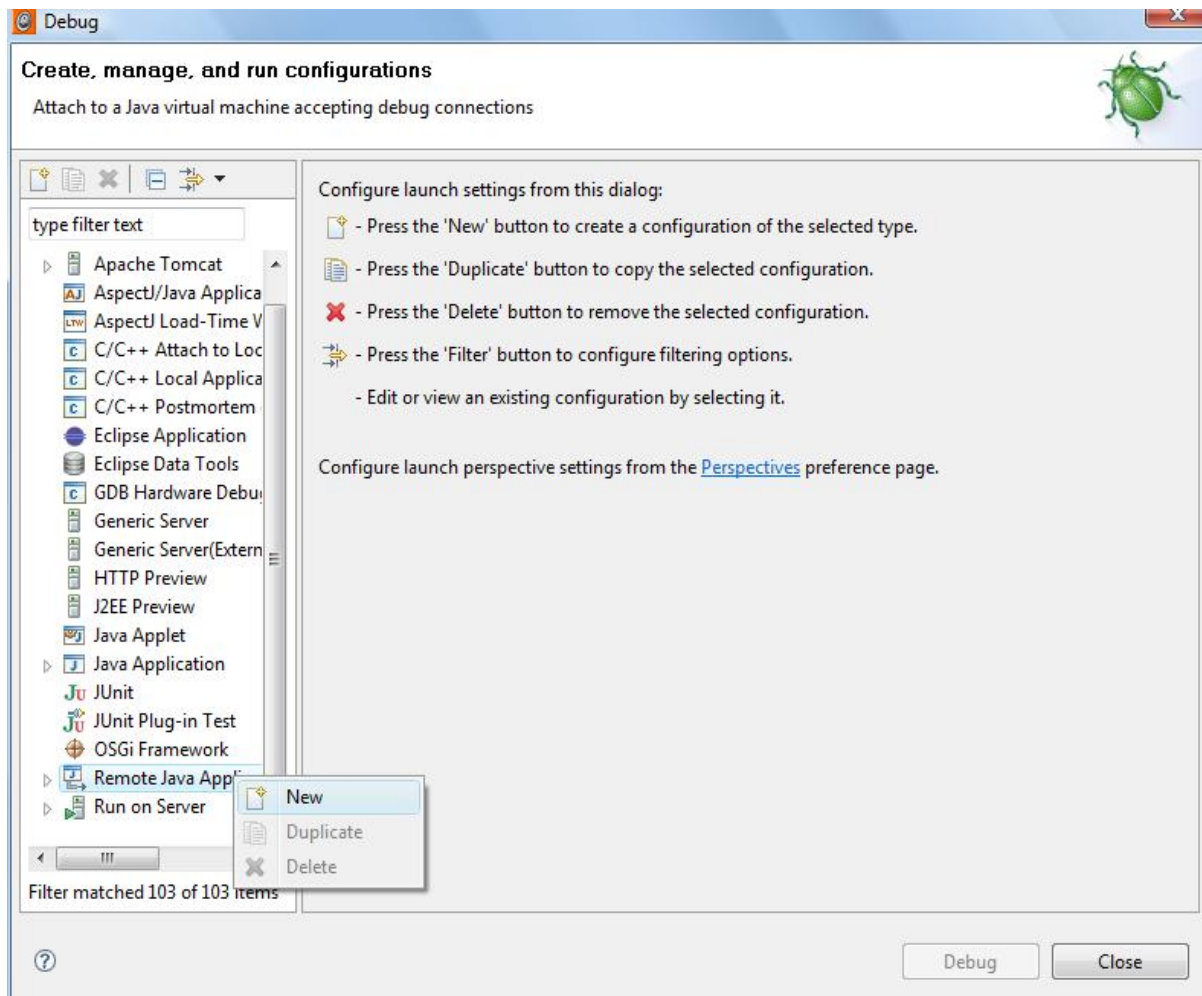
## Configuring Debug client

Steps for using Eclipse as a debug client:

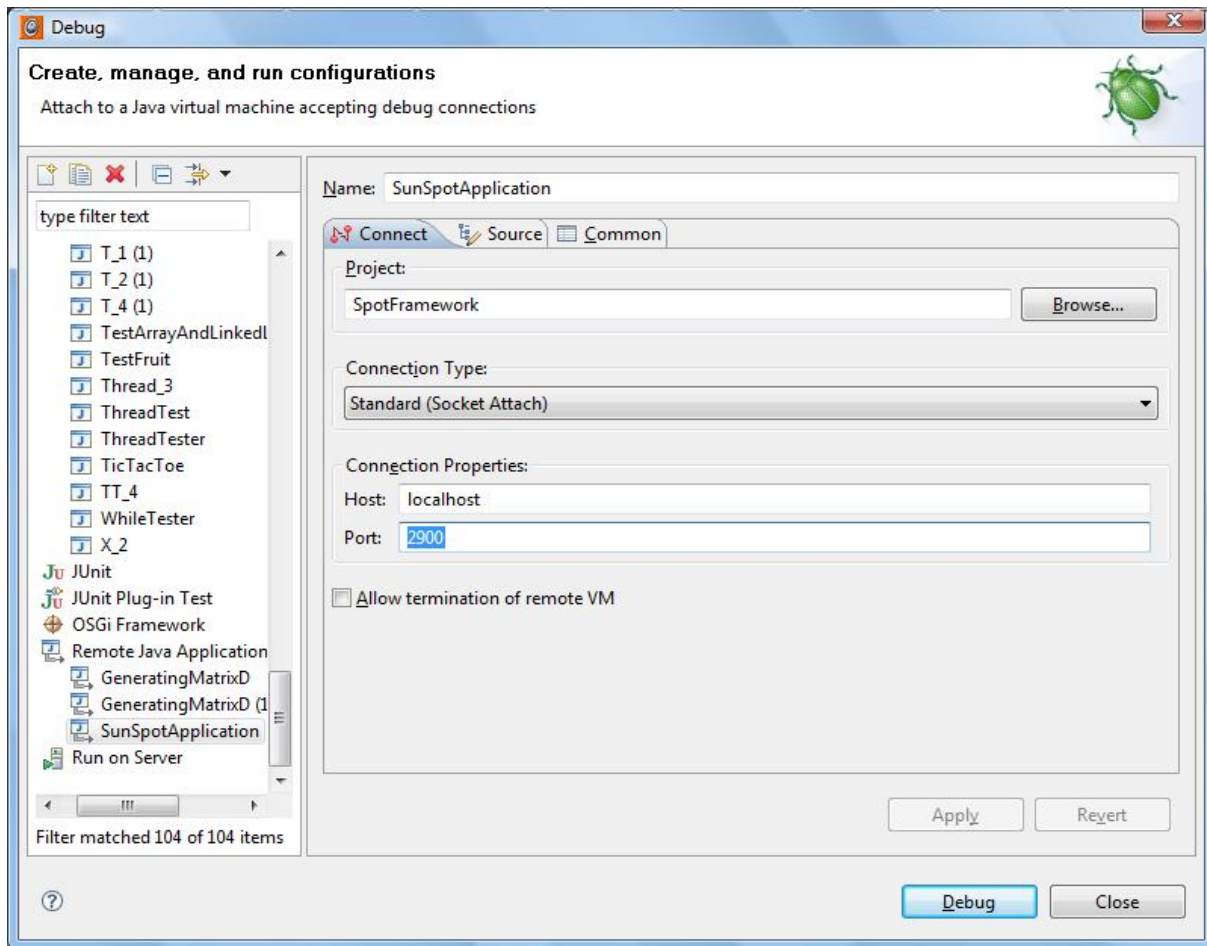
1. Go to Debug as, select Open Debug Dialog



## 2. Chose Remote Java Application and select “New”



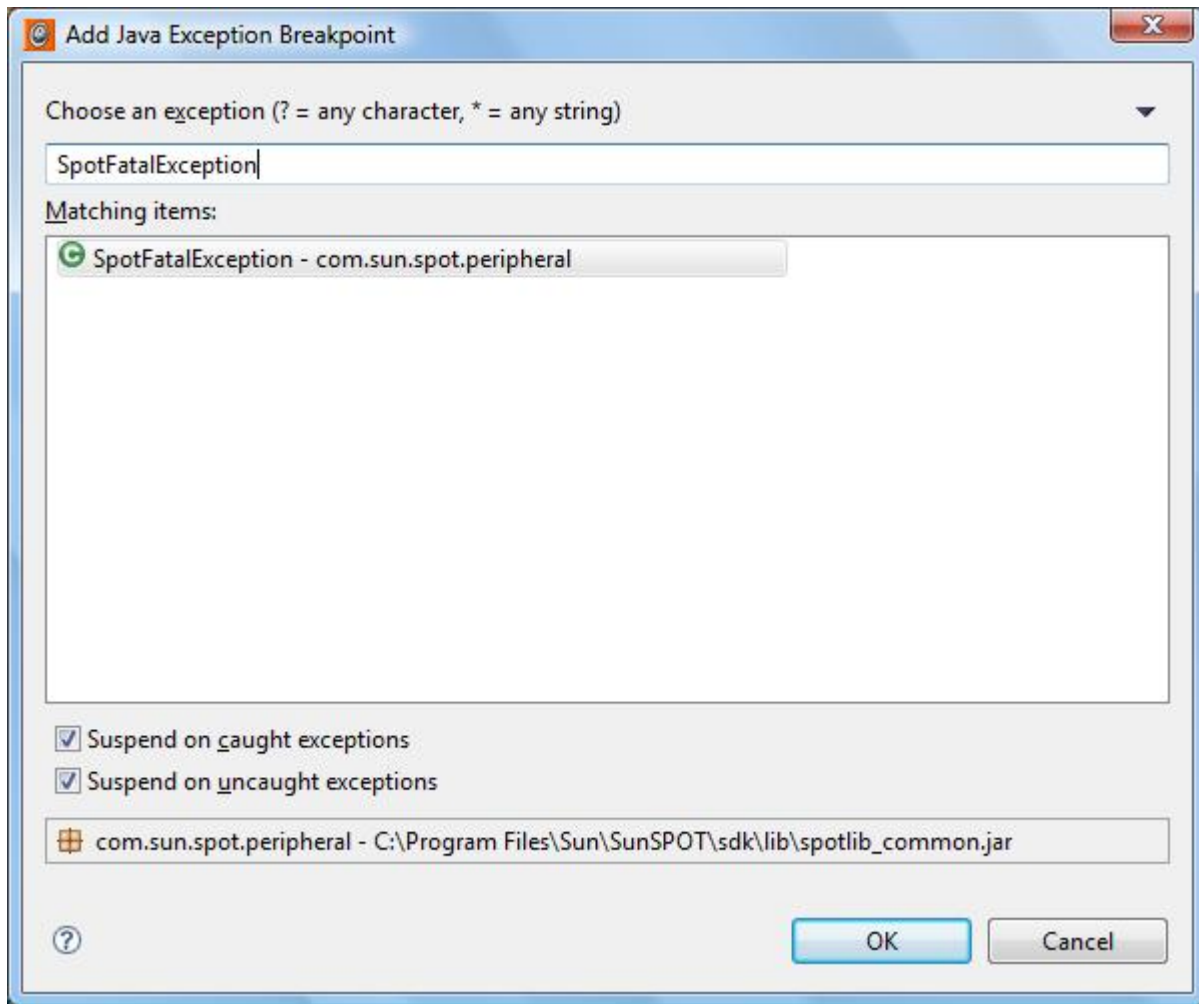
3. Choose port number as 2900. This is the port number where SDA and SDP will communicate remotely.



4. Goto Run and select Add Java Exception, select SpotFatalException

This exception class is used as a breakpoint, because applications which work correctly in host machine may break on the sun spots while debugging.

By setting the breakpoint at SpotFatalException, we get to know when the code breaks on the sun spots



## 4. Sun SPOT API

### Summary of the Sun Spots API

The Sun Spots API is the high level abstraction of the different components of the SPOT. The SPOT API covers three different aspects of the Device.

Squawk Virtual Machine Libraries

SPOT and Sensor Board Libraries

SPOT Generic Connection Framework

### Squawk Virtual Machine Libraries:

These libraries usually contain the modules required to run a J2ME application on the SPOT. SPOT based applications generally make use of the standard Java based libraries like java.io, java.lang, java.util and the Microedition based input output libraries. This library also provides the ability to directly address the Flash Memory and the other Operating System parameters.

### SPOT Sensor Board Libraries:

These libraries provide ways to access the different peripherals connected on embedded on the SPOT. One of the most important libraries is the com.sun.spot.peripheral which provides the high level abstraction of peripherals such as Parallel I/O , Timers, Power Manager, Flash Memory and LEDs. Sensor Related Objects can be accessed from the com.sun.spot.sensorboard.io . These include Objects that represent the Input Pins, Output Pins and Temperature Inputs on the Sensor board.

The com.sun.spot.sensorboard.peripheral allows the programmer to access the other peripheral chips that are connected to the sensor board. These include the Accelerometer, Light Sensor, Switch, Tone Generator, Tri Color LEDs etc.

### Generic Connection Framework:

This is the framework gives the programmer access to the Radio Connection API. This API enables the Host Application and the SPOT application to communicate using the Radio capabilities of the SPOT. The libraries that are generally used for connectivity and data transfer include : **com.sun.spot.io.j2me.radiogram**, **com.sun.spot.io.j2me.radiostream**.

In addition to wireless connection frameworks, we also have the ability to perform

Serial Communication using the USB port. The `com.sun.squawk.io.j2me.serial` library enables the SPOT to communicate with the Host application using the USB connectivity between the SPOT and the Host (PC)

These are the different APIs available for accessing the various SPOT based functionalities and Hardware.

The following section will cover some of the functionalities and interfaces in greater detail.

## **SPOT Sensors and Radio Communication:**

### **Sensor Board : EDemoBoard:**

This object provides access to the various peripherals and accessible components within the sensor board. This object is imported from the package `com.sun.spot.sensorboard`.

Most of the accessor methods of this object return objects of the sensors, that can be used to collect data. This board also provides an interface to the Flash EP ROM. This object is the most used object if one needs to access the sensor and I/O capabilities of the Board.

As the EDemoBoard is used as a Singleton, hence we never create an Instance of this object directly. The instance of this object is accessed using the **`getInstance()`** method

### **Sensors and other Peripherals:**

#### ***TriColorLed***

The Tricolor Led is a group of 8 LEDs that can be programmed to emit lights of different shades of RGB. This is generally used in-order to provide a visual notification of the internal processes or to indicate the progress in a certain operation.

The TriColorLED can be accessed using the package `com.sun.spot.sensorboard.peripheral`. We get an array of objects of type `ITriColorLED` when one performs the following operation on **`EDemoBoard.getInstance().getLEDs()`**. This array is used to set and get individual LEDs among the 8 available ones on the board.

The color of each LED can be set to a specific RGB value using the method `setRGB(int r,int g, int b)` where r,g,b range from 0 to 255. The object has `setOn()` and `setoff()` functionalities to turn the LED on and Off respectively.

One can also get the current value of R, G and B using the `getRed()`, `getGreen()` and `getBlue()` functionality. This can be used when multiple threads need to identify the current progress of any operation. The LED could be used a shared object for multiple instances or threads.

### ***Light Sensor***

The `LightSensor` object `ILightSensor` can be imported from the `com.sun.spot.sensorboard.peripheral` package. An instance of the Light Sensor can be obtained from the Sensor Board as `EDemoBoard.getInstance().getLightSensor()`. This object has two accessor methods `getRange()`- which gives the integer range of the input values and `getValue()` which gives the integer value of the sensor reading. External Light Sensors can be accessed using the `LightSensor` Object. The Constructor of this object takes in arguments for the pin the sensor is connected to , the range mask and the ADC controller to use for the `LightSensor`

### ***Temperature Sensor***

The `Temperature Sensor` object `ITemperatureInput` can be imported from the `com.sun.spot.sensorboard.io` package. Apart from the usual accessor methods `getValue` and `getRange()`. This object can be used to directly get the value of temperature in Celsius and Fahrenheit using the methods `getCelsius()` and `getFahrenheit()`. The following is the simple code for conversion from raw value to Celsius and Fahrenheit  
Celsius =  $(\text{getValue()} - 1024) / 4$  if the `getValue()` returns number  $\geq 512$  and  $\text{getValue()} / 4$  otherwise.  
Fahrenheit =  $(9.0 / 5.0) * \text{getCelsius()} + 32.0$

### ***3D Accelerometer***

The built-in 3D Accelerometer is imported as;

```
com.sun.spot.sensorboard.peripheral.LIS3L02AQAccelerometer.
```

An instance of the Object is obtained as `EDemoBoard.getInstance().getAccelerometer()`; This Accelerometer can measure using the scales 6G and 2G where G is the acceleration with respect to the Earth's Gravitational force. The accelerometer reports a voltage to indicate the acceleration along each axis. To convert this voltage to G's requires subtracting the zero acceleration voltage and then dividing by a gain constant:  
Acceleration in G's =  $(\text{Voltage} - \text{zero offset}) / \text{gain}$  A standard zero offset and gain are used by default, but since the accelerometer in each SPOT can differ by up to 10% from

the default it is best to calibrate each SPOT and save the offset and gain for each axis and scale.

The Offsets of the Accelerometer readings are obtained using the function call `setRestOffsets()`. This method calculates the current value of the accelerations across the various dimensions in both the 2G and 6G mode and stores the value. This method takes approximately 1 second to complete.

These rest values are used when calculating the relative acceleration across the x, y and z co-ordinates in the functions `getRelativeAccelZ()`, `getRelativeAccelY` and `getRelativeAccelX()`.

In addition to acceleration values, it is also possible to get the Tilt values in radians. This value indicates the tilt of the SPOT from each of the axes the range for these methods is from  $-\pi/2$  to  $+\pi/2$ . The methods that are used to find this are `getTiltX()`, `getTiltY()`, `getTiltZ()`.

The scale can be set to either 2G or 5G using the `setScale()` method.

### **Radio Communication:**

Communications over the radio usually take place using either datagrams or datastreams. These are analogous to UDP and TCP used for IP based networking. The datagram communication takes place with the help of the `RadiogramConnection` and `Radiogram` objects. Stream based connection takes place using the objects `RadioStreamConnection`.

### **Connections:**

#### **Datagram based connection :**

To listen and broadcast information the connection is opened as

`(RadiogramConnection)Connector.open("radiogram://:" + BROADCAST_PORT)` and `(DatagramConnection)Connector.open("radiogram://broadcast:" + CONNECTED_PORT)`

respectively. The Broadcast port can be arbitrarily decided. For peer to peer communication the connection is opened as

`(RadiogramConnection)Connector.open("radiogram://" + spotAddress + ":" + CONNECTED_PORT)`; where the `spotAddress` is the MAC address of the SPOT we want to connect to in the Long format.

#### **Stream based connection:**

A Radio Stream based connection is opened as `StreamConnection conn =`

`(StreamConnection) Connector.open("radiostream://destinationAddr:portNo")`. Streams

are then opened as `DataInputStream dis = conn.openDataInputStream();`  
`DataOutputStream dos = conn.openDataOutputStream();` and then stream based operations are performed on them, as in case of the standard Java IO Streams.

### ***Addressing:***

One can switch between the Long representation of the MAC address and the dotted hex representation. We call `IEEEAddress ieeeAddr = new IEEEAddress(addr)` where `addr` is the MAC address in the string form, and then `ieeeAddr.asLong()` is used to get the address of the SPOT in the long format. The corresponding address can be obtained from the long format by using `ieeeAddr.asDottedHex()`. Note: we can also pass the long form to the constructor of `IEEEAddress` to obtain an instance.

### ***Sending and Receiving data:***

#### **Datagrams:**

In case of datagrams the individual data packets are obtained as `Datagrams`. A `Datagram` is initialized by doing  
`(Datagram)conn.newDatagram(conn.getMaximumLength());`  
The connection then receives onto the datagram using the method  
`conn.receive(Datagram object instance);` The datagram object consists of several accessor methods to get the data in various formats like `readShort()`, `readByte()` etc. The datagram object can be written into by using similar methods like `writeByte()`, `writeShort()`, `writeUTF()` etc. This datagram is sent over the network using the method  
`conn.send(Datagram Object Instance);`

#### **Streams:**

In case of streams the data is sent and received as normal Input and Output Streams in java.

#### ***Timeouts:***

The connections can have a specific timeout associated between them, so as to not time out the connection if no data is received or sent in a specific time period.

### ***LinkQuality and Received Signal Strength Indication:***

The `Datagram` objects can be used to identify the Link Quality and the RSSI from the datagram. Link Quality is obtained from Correlation parameters and has a value between 0 and 255. 0 indicating bad link quality and 255 indicating good. The RSSI It ranges from +60 (strong) to -60 (weak). To convert it to decibels relative to 1 mW (= 0 dBm) we, subtract 45 from it.

The `getRssi()` is used to get the Rssi of the link, `getLinkQuality()` is used to get the link quality of the link. `getCorr()` is used to get the average correlation value of first 4 bytes of the packet header (~110 to indicate good correlation and ~50 to indicate low correlation).

These are the most frequently used interfaces and objects that are used for communicating with the Sensors and perform Radio Operations. There are a multitude of other interfaces and objects available in the API that allow us to access other peripherals and parameters of existing and external peripherals. A brief explanation of their utility can be found on the Javadocs page of the Sun SPOTs Homepage. Please check the Sun SPOT V2.0 (Orange) API at <http://www.sunspotworld.com/docs/Orange/javadoc/> for more details about the API.