# Using the SPOT Emulator in Solarium

Ron Goldman

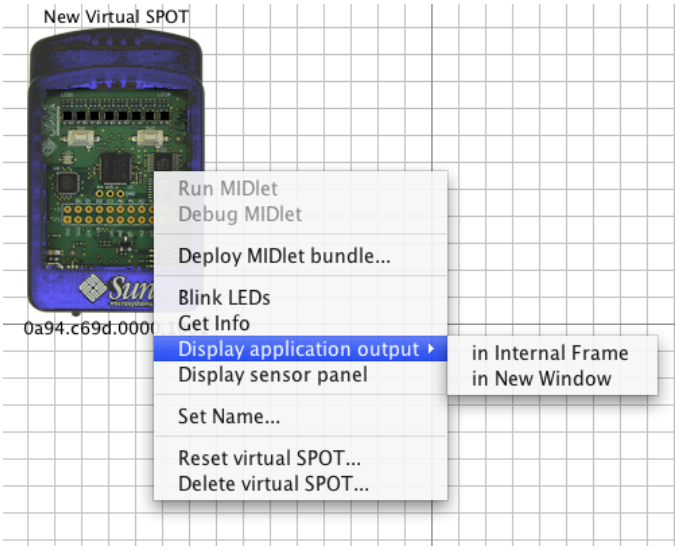
June 2008

## Summary

Solarium includes an emulator capable of running a Sun SPOT application on your desktop computer. This allows for testing a program before deploying it to a real SPOT, or if a real SPOT is not available. Instead of a physical sensorboard, Solarium displays a virtual SPOT with a control panel where you can set any of the potential sensor inputs (e.g. light level, temperature, digital pin inputs, analog input voltages, and accelerometer values). Your application can control the LEDs' color that is displayed in the virtual SPOT image, just like it would a real SPOT. You can click with the mouse on the push button switches in the virtual SPOT image to press and release the switches. Receiving and sending via the radio is also supported. Each virtual SPOT is assigned its own address and can broadcast or unicast to the other virtual SPOTs. If a shared basestation is available a virtual SPOT can also interact over the radio with real SPOTs.

## How to run an application in a virtual Sun SPOT

The first step is to start up a version of Solarium on your computer. From any SPOT project folder the command **ant solarium** will do so. Alternatively you can start Solarium using the SPOTManager tool. Once Solarium is running make sure that a graphic *Grid View* is displayed (**View** > **Grid View**). Then from the **File** menu select the command **New virtual SPOT**. This will display an image of a Sun SPOT. If you right-click on the virtual SPOT you will see a menu of possible commands.




The command **Set Name…** allows you to give the virtual SPOT a descriptive name to help you identify it. Each virtual SPOT has a label above it with its name and another label below it with its IEEE radio address.

The **Deploy MIDlet bundle…** command lets you *deploy* a SPOT application to the virtual SPOT. It will bring up a file chooser dialog that you can use to navigate to a SPOT project directory. You can select an existing jar file created with the **ant jar-app** command or the project's build.xml file, in which case a process will be spawned to compile the source code, build the jar file, and then load it. Once you have loaded some MIDlets you can use the **Run MIDlet** command to display a submenu listing all of the MIDlets contained in the deployed jar file and allow you to start up whatever one you want. Any running MIDlets will be displayed in a box to the right of the virtual SPOT. Clicking on a running MIDlet will display a popup menu that lets you tell the MIDlet to exit.

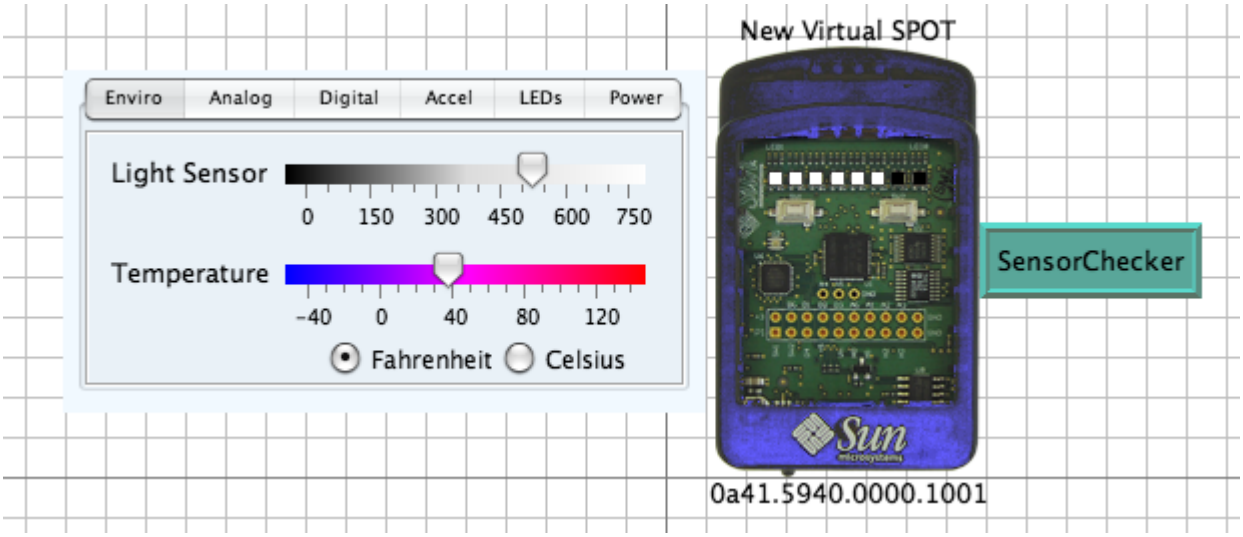
There is also a **Debug MIDlet** command that will allow you to connect an external Java Debugger to a MIDlet and debug it.

To try the Emulator out please use the **Deploy MIDlet bundle…** command to load in the *emulator_demo.jar* file located in the *Demos/EmulatorDemo* folder. Once it is loaded run the *Sawtooth* MIDlet. As it runs you will see the LEDs of the virtual SPOT be turned on, one by one, each brighter than the previous, until all are lit at which point they are all turned off and the cycle repeats. Right click on the Sawtooth application box and exit it.

The **Reset virtual SPOT…** command will cause any running MIDlets to be killed and the Squawk VM to be restarted. If a jar file had been specified earlier, then it is automatically reloaded and you can run any of the MIDlets defined in it.

You can use the **Display application output** command to display a new window where anything printed by the SPOT application to System.out or System.err will be displayed. This new window can be an Internal Frame that is displayed beneath the virtual SPOT in the main Solarium window, or it can be in a New Window. If you reset the virtual SPOT you will see messages printed when the old Squawk VM exits and the new one is started up. If the output window is covered up the **Display application output** > **in New Window** command will bring it to the front.

To demonstrate how to change the various sensor values, run the *SensorChecker* demo from the already loaded *emulator_demo.jar* file. This application uses the LEDs on the virtual SPOT to display a value read from one of the SPOT's sensors. When it is started the light sensor reading is displayed in white. To change the light sensor value use the **Display sensor panel** command to bring up the sensor panel. On the leftmost *Enviro* tab are two sliders for controlling the value the SPOT will read for the light sensor and for the internal thermometer. As you move the light sensor slider left and right you will see the number of LEDs change appropriately.




Temperature is specified in degrees Fahrenheit or Celsius, light readings in the raw value returned from the A/D, analog inputs in volts, and acceleration in gravities (G's).

The *SensorChecker* demo has four different modes:

1. Display the light sensor reading in white
2. Display the temperature sensor reading in red.
3. Display the analog input A0 in green.
4. Display the Z acceleration in blue.

Push the left switch (SW1) by clicking on it with the mouse to advance to the next mode. Switch to the different tabs of the sensor panel to access the different sensors. As you move the slider for the current sensor you will see the LED display change.

If you go to the *Digital Pins* tab you will see the current mode shown by the application setting one of the high current output pins, H0-H3, to high. As you cycle through the different modes the SPOT application will change which pin is set to high. The digital input/output pins, D0-D4, are enabled when they are being used as an input so you can set their value. When they are being used as an output they are disabled and the SPOT application can set their value to low or high. For the *SensorChecker* demo D0 is an output, while D1-D4 are set as inputs. The application reads the value of D1 and then sets D0 to be the same. Try changing the value of D1 and watch as D0 is also changed.

Note: the popup menu for a virtual SPOT can also be used from the *Tree View*. From the *Tree View* the **Display sensor panel** command will create a new window to display the sensor panel. To locate a virtual SPOT in the *Grid View* one can cause its LEDs to blink using the **Blink LEDs** command from its popup menu in the *Tree View*.

The **Get info** command will bring up a new window giving some information about the virtual SPOT: its IEEE address, the jar file loaded (if any), and the names of all available MIDlets.

When you are done with the virtual SPOT it can be deleted using the **Delete virtual SPOT** command.

From the **Emulator** menu one can use the **Save virtual configuration...** command to write out a file that will store the state of all of the virtual SPOTs: each virtual SPOT's name and radio address, what jar file it is using, what MIDlets are running, and where the virtual SPOT is located on the grid. You can specify whether you want the current radio address kept for use when the configuration is read back in or whether you want a new address to be used. You can also specify whether or not to automatically restart any currently running MIDlets when the configuration is read in. Along with the configuration you can include a textual description that will be displayed when the configuration is reloaded. The **Open virtual SPOT...** command is the way to select a previously saved configuration file and reload it. When it is reloaded any descriptive text associated with it will be displayed in a new window. You can cause this window to be redisplayed at any time using the **Display virtual configuration description** command. Finally the **Emulator** menu has the **Delete all virtual SPOTs...** command to remove any virtual SPOTs currently defined in Solarium.

For an example of loading a predefined configuration do **Delete all virtual SPOTs...** followed by **Open virtual configuration...** and select the file *emulator_demo.xml* from the *EmulatorDemo* in the *Demos* folder. That will create 4 virtual Spots in Solarium, and start 3 of them running various demo apps. It will also display a window with a textual description of the available MIDlets.

Note: if you start Solarium from the command line you can specify a previously saved configuration file and have it automatically loaded when Solarium starts up. To do so just set the ant property `config.file` either on the command line (e.g. `"-Dconfig.file=<path to config file>"`) or in your project's `build.properties` file. For example:

```
cd SunSPOT/sdk/Demos/EmulatorDemo
ant solarium -Dconfig.file=emulator_demo.xml
```

**Using the Radio**

Virtual SPOTs can communicate with each other by opening radio connections, both broadcast and point-to-point. Instead of using an actual radio these connections take place over regular and multicast sockets.

When a basestation SPOT is connected to the host computer and a shared basestation is running, virtual SPOTs can also use it to communicate with real SPOTs using the basestation's radio. The advantage of using a shared basestation is that multiple host applications can then all access the radio. One disadvantage is that communication from a host application to a target SPOT takes two radio hops, in contrast to the one hop needed with a dedicated basestation. Another disadvantage is that run-time manipulation of the basestation SPOT’s radio channel, pan id or output power is not currently possible.

To always use a shared basestation add the following line to your *.sunspot.properties* file:

```
multi.process.basestation.sharing=true
```

Please note that some Linux distributions (e.g. SuSE) may not have multicasting enabled by default, which will prevent shared basestation operation and also any “radio” use by virtual SPOTs.

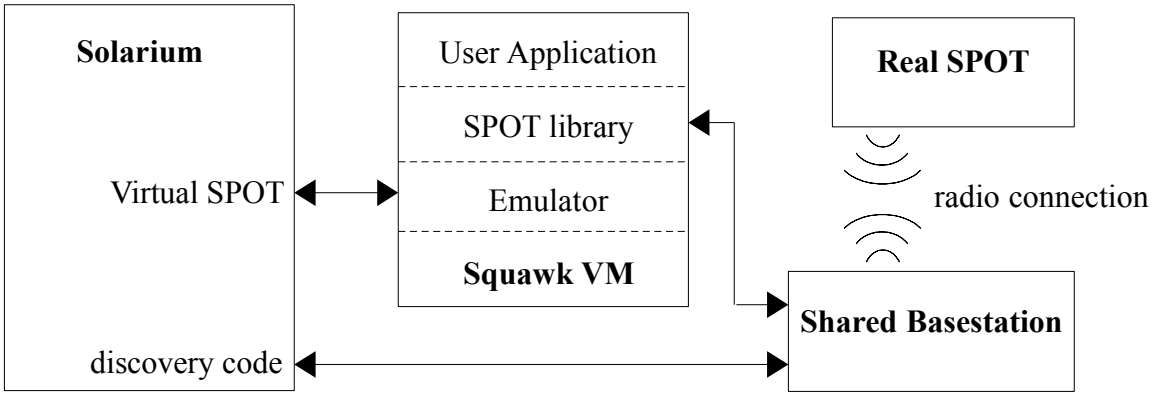
Note: virtual SPOTs can also communicate with SPOT host applications using the radio, but only if the host application is run with `multi.process.basestation.sharing` set to `true`.

The *EmulatorDemo* provides several sample MIDlets that use the radio. Start with the *BroadcastCount* demo which uses the left switch (SW1) to broadcast a message to set the color displayed in the LEDs of all receiving SPOTs and the right switch (SW2) to count in binary on the receiving SPOTs' LEDs. If a shared basestation is available then try deploying the *EmulatorDemo* to a real, physical SPOT and having it then interact with the virtual SPOTs via the radio.

**How the Emulator Works**

When you create a new virtual SPOT in Solarium, a new process is started to run the emulator code in a Squawk VM. The emulator code communicates over a socket connection with the virtual SPOT GUI code in Solarium. For example when the SPOT application changes the RGB value of an LED that information is passed to the virtual SPOT GUI code that updates the display for that LED with the new RGB value. Likewise when the user clicks one of the virtual SPOT's switches using the mouse, Solarium sends a message to the emulator code that the switch has been clicked, which can then be noticed by the SPOT application.

Here's a block diagram of the Emulator architecture:




Each virtual SPOT has its own Squawk VM running in a separate process on the host computer. Each Squawk VM contains a complete host-side radio stack as part of the SPOT library, which allows the SPOT application to communicate with other SPOT applications running on the host computer, such as other virtual SPOTs, using sockets or real SPOTs via radio if a shared basestation is running.

**Emulation vs Simulation**

The distinction between emulation and simulation is not always clear, so let me explain how I am using the two terms. When a computer application is "run" in an emulator, the emulator is mimicking the behavior of a different computer system, which allows the user program to run as if it were on that other system. While the important functionality is preserved, other aspects, such as the time to do a given operation, may be quite different. Hence a program may run much slower when it is emulated.

A simulation, in comparison, is built by creating a model of a system and identifying various properties that will be accurately modeled as to how their values change. There is usually some sort of abstraction involved, for example a weather simulation does not model individual molecules but rather breaks the world up into a grid of cells of various sizes (ranging from meters to kilometers) and then characterizes several parameters for each cell.

The current Solarium implementation is primarily an emulator since it actually runs a SPOT application in a Squawk VM, just like the VM on a real SPOT. Likewise radio interaction between virtual SPOTs is emulated with data sent via packets and streams from one (virtual) SPOT to another. Only the SPOT's interaction with the environment is simulated using a simple model where the user needs to explicitly set the current sensor values. Future versions may incorporate more simulation of SPOT properties like battery level or radio range.

**What's Missing?**

The initial version of the Emulator allows a SPOT application to control the LEDs and digital output pins, read various sensor inputs—switches, light level, temperature, digital input pins, analog input voltages, and accelerometer values—and send and receive radio messages. However there are other aspects of the Sun SPOT that are not currently implemented.

Like any SPOT host application using a shared basestation, a virtual SPOT cannot control the radio channel, pan id or power level. Nor is there the ability to turn the radio off and on.

Not available is various sensorboard functionality such as the UART, tone generation, servo control—including pulse width modulation (PWM), pulse generation, timing a pulse's width, and doing logical operations on the Atmega registers. These unimplemented features currently act as no-ops rather than throwing any exceptions.

There is currently no emulation of the low-level processor hardware functionality provided by the following classes and interfaces in the SPOT library: *ISpiMaster, IAT91_PIO, IAT91_AIC, IAT91_TC, IProprietaryRadio, I802_15_4_PHY, SpotPins, FiqInterruptDaemon, ISecuredSiliconArea, ConfigPage, ExternalBoardMap, ExternalBoardProperties, IFlashMemoryDevice, ISleepManager, ILTC3455, IUSBPowerDaemon, IAT91_PowerManager, IDMAMemoryManager,* and *OTACommandServer*. Nor is there support for saving persistent properties, getting the SPOT's public key or reading the current system tick. Attempts by an emulated SPOT application to use these unimplemented features will cause a *SpotFatalException* to be thrown.

Also missing is the Java ME record management system (RMS) functionality.

**Future Directions**

While there is no schedule for when additional features will be added to the Emulator here are some likely areas for improvement in the not too distant future.

Currently no statistics or metrics are kept. Instrumenting the virtual SPOT's software stack should make it possible to report on an applications activities such as radio usage, idle time, memory usage (including number of GCs needed), etc.

The current architecture of the Emulator makes it fairly easy to add code to Solarium that can dynamically generate sensor readings for a virtual SPOT. This may take the form of a palette of virtual devices, such as a signal generator that could be hooked up to an analog input, or a way to load user written Java code into Solarium and have it control the input to a virtual SPOT, e.g. to compute the acceleration on the SPOT as it is moved.

Some more ambitious possible future directions involve adding simulation of the radio—controlling the topology of radio connections, varying the percentage of dropped packets, setting transmission power levels, or making transmissions visible in the Emulator so as to visualize the radio traffic—or of the power controller—simulating battery usage.