Solarium User's Guide

Red Release 5.0

Sun Labs
July 2009

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300
Document Revision 1.0.0

Table of Contents

Solarium User's Guide

Solarium is a Java™ application that can be used to remotely manage a network of Sun SPOTs. With Solarium, you can discover nearby SPOTs and manage the complete life-cycle of applications running on those devices. Here is some of what Solarium has to offer:

Discovering and displaying SPOTs – Solarium can discover and display SPOTs that are connected to the desktop via USB or can be reached via radio communication.

Interacting with SPOTs – Solarium can be used to load and unload software on a SPOT, start/pause/resume/stop applications and query the current state of a device, e.g. memory usage, and energy statistics. A Radio View provides a way to visualize the radio connectivity between SPOTs. Using a Deployment View makes it easier to describe and manage a group of Sun SPOTs.

Managing a Network of SPOTs – Solarium provides a special *Deployment View* to make it easier to manage a network of Sun SPOTs. A deployment specifies what application should be loaded on each SPOT and allows you to deploy those applications with a single button press. The Deployment View also shows the current status of each SPOT. A deployment can also specify any associated host applications that need to be run.

Emulating SPOTs – Solarium also includes an emulator that can be used to run applications on a virtual Sun SPOT. The new Robot View enables a virtual SPOT to control a virtual iRobot Create and explore several different physical environments.

Discovering and Displaying SPOTs

Solarium can detect and display SPOTs that are connected to the desktop via USB either directly or through one or more USB hubs. It can also detect nearby SPOTs via wireless communication.

Turning on the OTA Command Server

The "over-the-air" (OTA) command server is a piece of software that runs on the SPOTs and listens for management commands sent wirelessly from a host application such as Solarium.

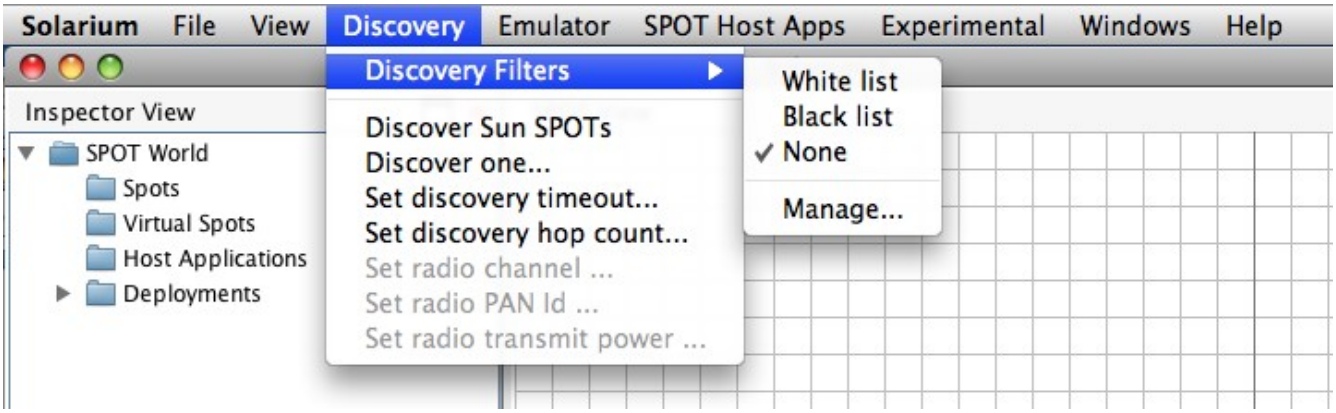
SPOTs must have their OTA command server turned on to be discoverable via wireless communication. The OTA command server is turned on by default. If it is turned off for some reason (as indicated by `ant system-properties` showing `spot.ota.enable: false`), it can be turned on again by connecting the SPOT to the desktop via USB and using any one of the following options:

• Issuing the `ant enableota` command in a terminal window.

• Using the "Enable OTA" button in SPOT Manager

Controlling SPOT Discovery

The **Discovery** pull-down menu in the main Solarium menu bar allows you to control the discovery process that Solarium uses to find the local SPOTs. The choices are:



• Discovery Filters

This command allows the discovered SPOTs to be filtered according to a white (include) or black (exclude) list. It also allows one to edit which SPOT addresses go on the white or black lists.

• Discover Sun SPOTs

This choice sends a broadcast discovery message, asking Sun SPOTs to identify themselves. If the broadcast message reaches a Sun SPOT running the OTA command, that SPOT will respond with basic information, e.g. its IEEE address, the version of its SDK, etc. This command also causes Solarium to

scan the USB ports on the host and discover any connected SPOTs (even if they have their OTA command server turned off). Discovered SPOTs are displayed in Solarium.

• Discover one...

This choice allows you to send a unicast discovery message to a specific IEEE address. Solarium will attempt to find that particular SPOT. The broadcast hop count setting described below is ignored when the discovery message is sent as a unicast.

• Set discovery time out...

This choice specifies how long Solarium will wait for an answer to its broadcast discovery message. It defaults to three seconds.

• Set discovery hop count...

This choice specifies how many radio hops a broadcast discovery message may traverse before it is no longer forwarded from SPOT to SPOT. When using a shared basestation, this setting must be at least two, since the communication between the host application and the shared basestation counts as one hop. Therefore, at least two hops are required to reach any actual SPOTs. If you want Solarium to reach SPOTs beyond the direct radio range of the basestation, the hop count should be set to three or more.

• Set radio channel...

Allows you to specify the radio channel used for discovery. Defaults to 26.

Note: It is not possible to change the radio channel when using a shared basestation.

• Set radio PAN Id...

Allows you to specify the PAN ID used by Solarium during discovery. Defaults to 3.

Note: It is not possible to change the PAN ID when using a shared basestation.

• Set radio transmit power...

Allows you to specify the radio transmit power used by Solarium during discovery. Defaults to full power (0).

Note: It is not possible to change the radio transmit power when using a shared basestation.

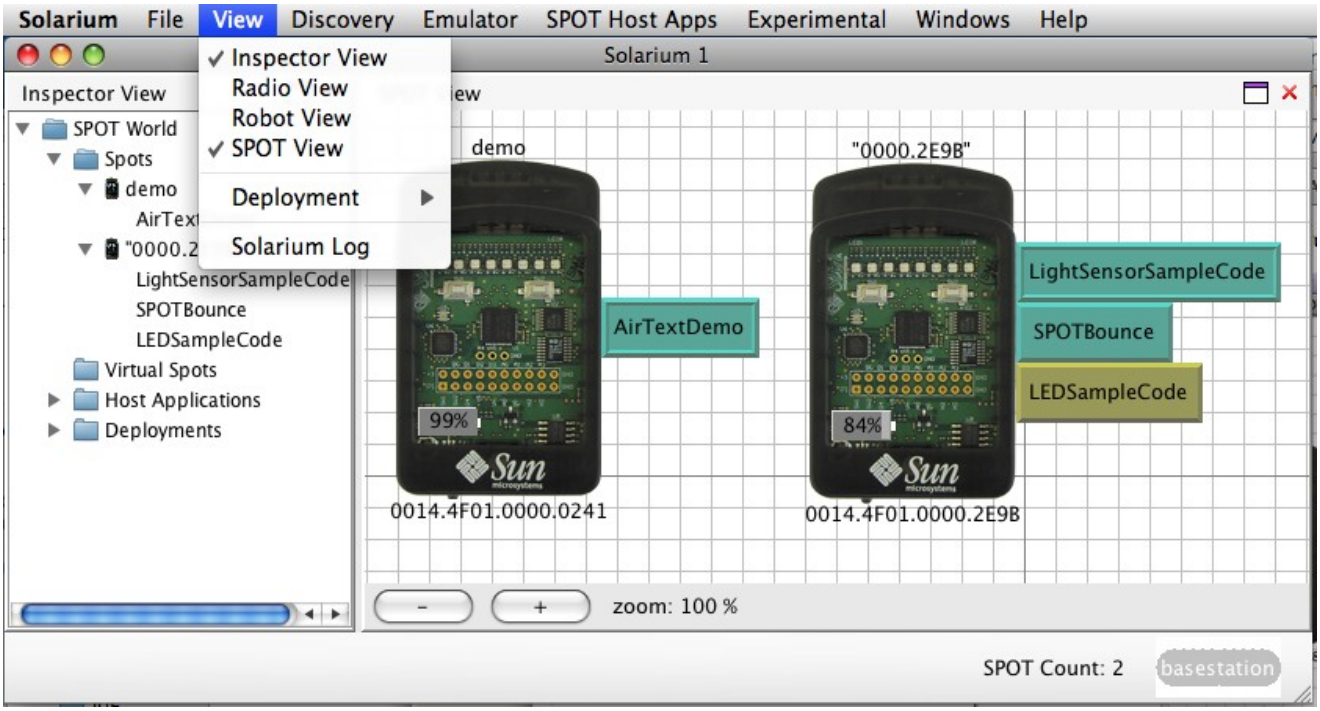
Controlling Which SPOTs are Displayed

By default, Solarium displays all of the SPOTs it discovers. In an environment with lots of SPOTs, a user might want to ignore some of them to avoid cluttering the Solarium display unnecessarily. The .solarium.properties file in the user's home directory allows one to specify either a "blacklist" or a "whitelist" of SPOTs. If a whitelist is specified, only the SPOTs on that list are displayed, all others are ignored. If a blacklist is specified, SPOTs on that list are ignored and all others are displayed. To edit these lists, simply select the “manage...” item from the “Discovery > Discovery Filters” menu. If you wish to edit these lists without using the tool, you can (carefully) change your .solarium.properties file. For example, to ignore all SPOTs except those with the IEEE address of 0014.4F01.0000.020B or 0014.4F01.0000.048D insert the following line in your .solarium.properties file:

<entry key="spots.whitelist">0014.4F01.0000.020B,0014.4F01.0000.048D</entry>

Controlling How SPOTs are Displayed

SPOTs can be displayed in several different ways by Solarium. By default Solarium starts up displaying a tree-like *Inspector View*, shown here to the left, and a 2-dimensional *SPOT View*, shown here against the quadrille background to the right. The views exposed are controlled through the **View** pull-down menu on the menu bar. Both these views also show the applications running on each SPOT.



After Solarium discovers the SPOTs, it will display all of them along with the running applications currently running on those devices.

The *Deployment View*, *Radio View* and *Robot View* will be described below.

Note: Multiple views can be combined (tiled) in a single window. Just grab and drag the view's title bar, e.g. where it says *Inspector View*. Views can be dragged into an existing window, or dragged out onto the screen to make a new window.[1]

Interpreting Visual Cues

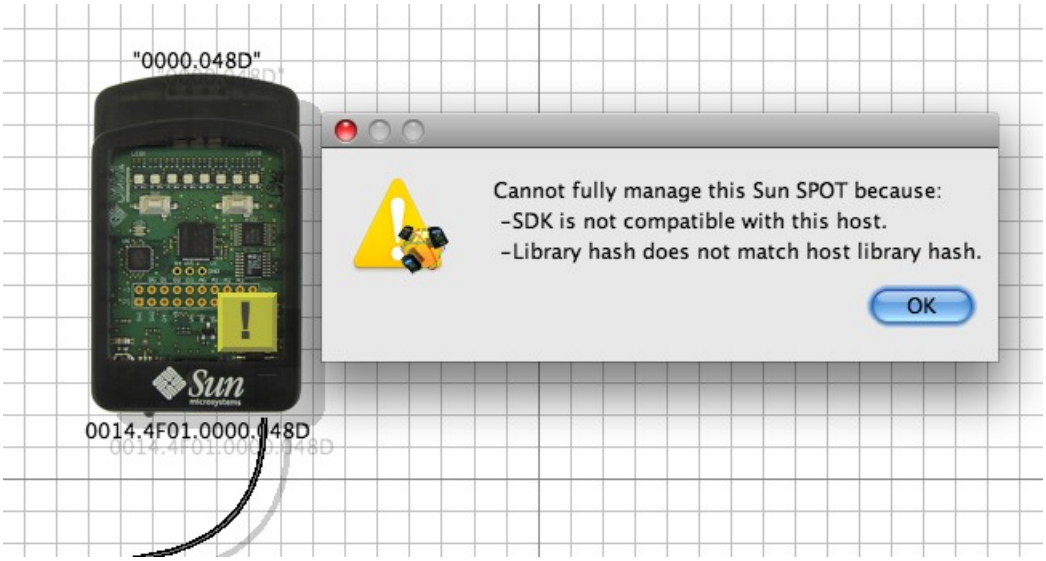
Solarium uses a number of visual cues to indicate status information about the SPOTs:

•USB-connected SPOTs are drawn with a *tail* to indicate a USB cable.

•A yellow warning rectangle on a SPOT indicates that Solarium cannot fully manage that SPOT. Clicking on the yellow sign will bring up a panel listing all of the reasons, e.g., that SPOT may have a different owner or may have a version of the SDK that's different from the version under which Solarium is running.

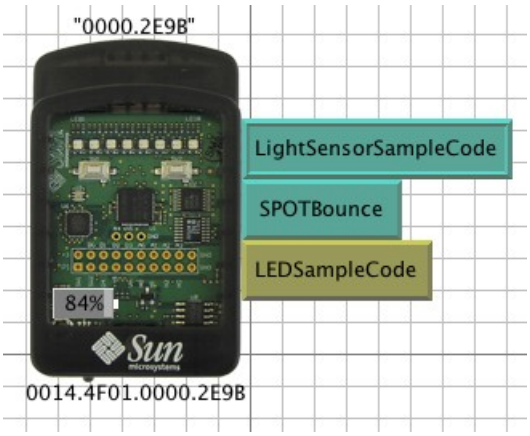
1. Thanks to Benjamin Sigg, the creator of DockingFrames for his open source Java Swing docking framework, which is available from http://dock.javaforge.com/.

The figure below shows a USB-connected SPOT that cannot be fully managed for the reasons shown.



•An application running in the master isolate is drawn as greenish-blue rectangle with a raised bezel while an application running in child isolate is drawn as a greenish-blue rectangle with a plain bezel. An application that is currently paused is drawn as an muted yellow colored rectangle. Only applications running in a child isolate can be paused.

In the figure below, the *LightSensor* application is running in the master isolate, the *SPOTBounce* application is running in a child isolate and the *LEDSampleCode* application is paused.



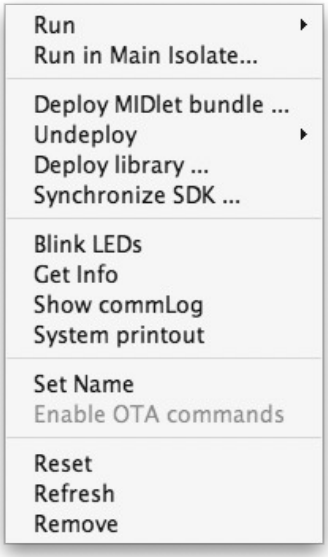
- Remaining battery power is shown in the small gray rectangle drawn in the bottom, left of the SPOT. In this case the SPOT has 84% of its battery capacity remaining.

Interacting with SPOTs

This section highlights the various ways in which a user may interact with SPOTs. Refer to the Emulator documentation on how to interact with virtual SPOTs.

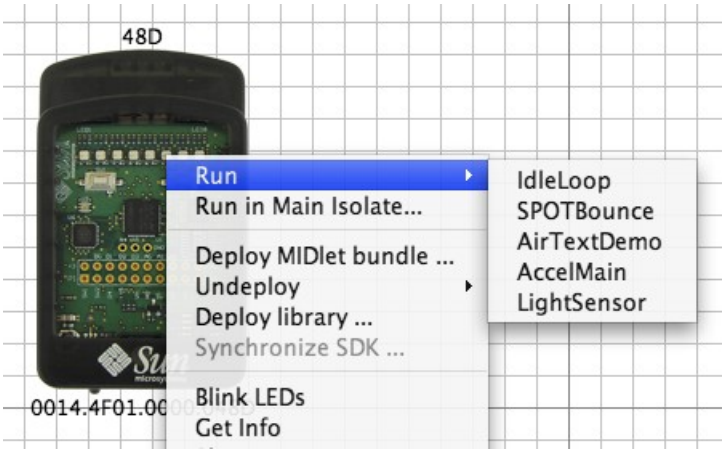
Manipulating SPOTs

If you right-click on a SPOT, the following menu will display. Some of the menu items are only enabled for USB-connected SPOTs while others require a SPOT to have its OTA command server enabled. Disabled menu items are displayed in gray.



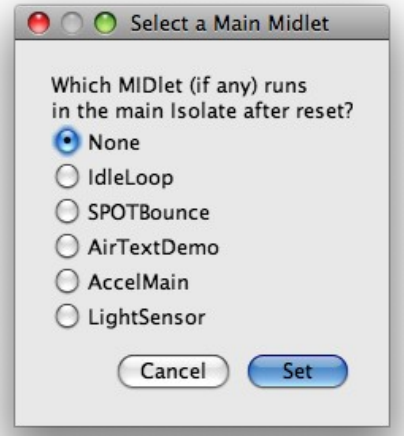
• Run

This menu option brings up a sub-menu listing all installed applications and selecting any application on this list starts it running.



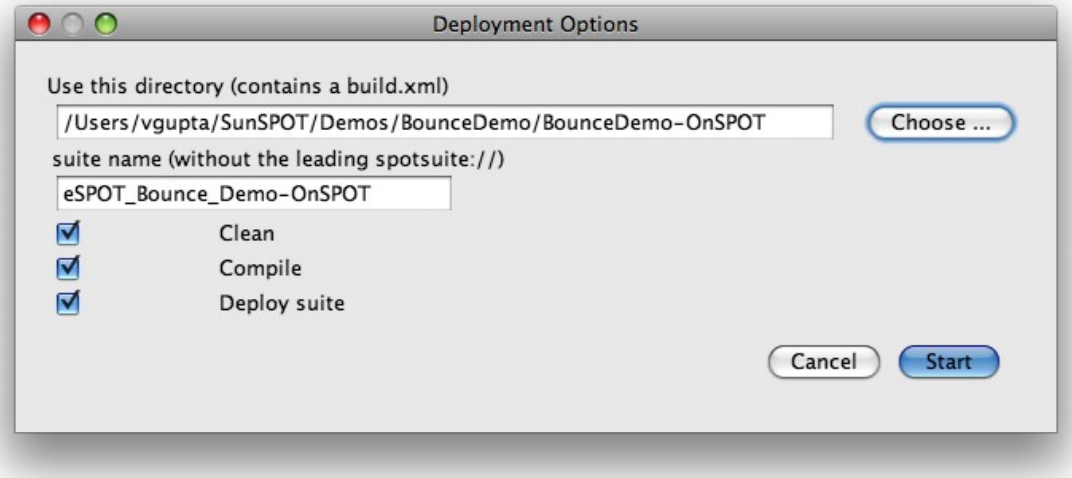
• Run in Main Isolate

This menu option brings up a panel where the user can choose which application (if any) to run in the main isolate. Only one application may run in the main isolate and the SPOT must be reset for this change to take effect. After the reset, the only application running on the SPOT would be the chosen one, all other previously running applications will be terminated.



• Deploy MIDlet bundle ...

Allows you to specify a project to be deployed to the selected Sun SPOT. The following dialog box will display:



By default, the project directory will be cleaned, all source code will be recompiled and the resulting suite will be installed on the SPOT. The clean and compile steps may be skipped by unchecking the appropriate boxes. This might make sense, for example, if the application suite was created recently and the same suite is being deployed on multiple SPOTs.

• Undeploy

This menu option brings up a sub-menu listing all installed suites and selecting any suite on this list will unload it from the SPOT. The command will fail if the suite is in use, e.g., one of the applications in the suite is currently active. If a suite includes multiple applications, all of them will be unloaded together.

• Deploy library...

Downloads to the SPOT a fresh copy of the current library (as specified by the spot.library.name property in the .sunspot.properties file in the user's home directory) on to the SPOT in the active SDK. After the library is installed, the SPOT will be reset and all previously running applications terminated.

This command is useful, for example, when a user has created a new library using the ant library command. Refer to the *Sun SPOT Developer's Guide* for details on how to rebuild the library.

• Synchronize SDK...

Downloads system software from the Sun SPOT SDK currently active on the host (where Solarium is running) to the selected SPOT. This option is only enabled for USB-connected SPOTs.

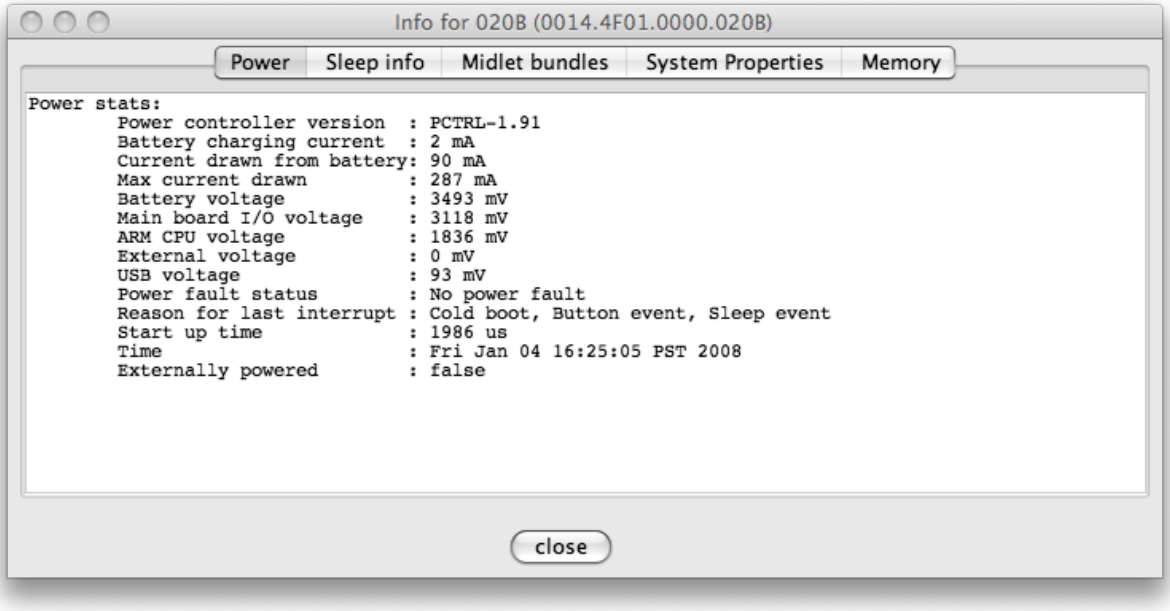
• Blink LEDs

Causes the SPOT to blink its LEDs. This command is useful for locating the physical SPOT corresponding to the SPOT on display.

On SPOTs equipped with the eDemo sensor board, all of its eight tricolor LEDs as well as the Activity LED on the main board are flashed. If a SPOT does not have the eDemo sensor board installed, only the activity LED is flashed. Refer to the *Sun SPOT Owner's Manual* for the location of the Activity LED.

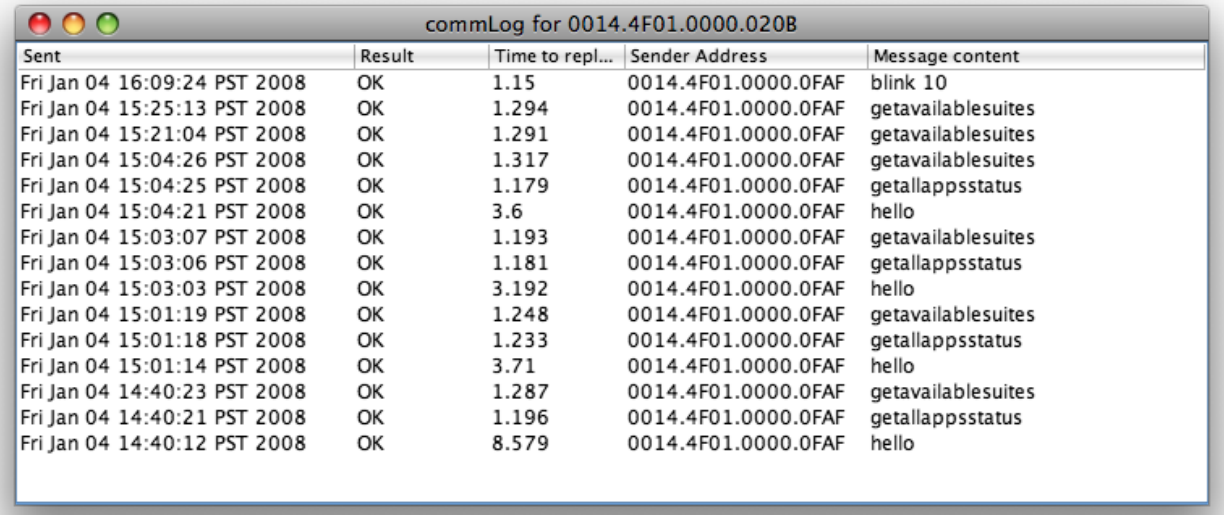
• Get Info

Opens a window containing multiple tabs. Each tab has information on a different aspect of the selected Sun SPOT, e.g., installed applications, system properties, and energy/memory statistics.



• Show commLog

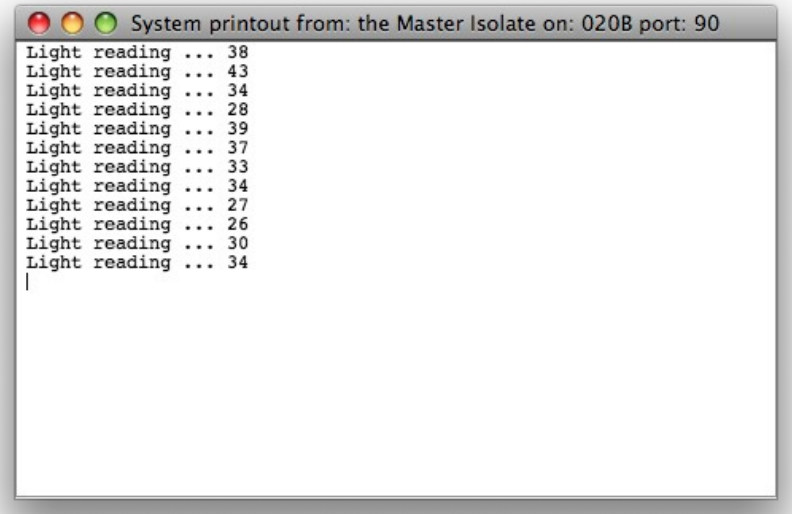
Opens a window showing a log of all the OTA commands issued to the selected SPOT.



Sent	Result	Time to repl...	Sender Address	Message content
Fri Jan 04 16:09:24 PST 2008	OK	1.15	0014.4F01.0000.0FAF	blink 10
Fri Jan 04 15:25:13 PST 2008	OK	1.294	0014.4F01.0000.0FAF	getavailablesuites
Fri Jan 04 15:21:04 PST 2008	OK	1.291	0014.4F01.0000.0FAF	getavailablesuites
Fri Jan 04 15:04:26 PST 2008	OK	1.317	0014.4F01.0000.0FAF	getavailablesuites
Fri Jan 04 15:04:25 PST 2008	OK	1.179	0014.4F01.0000.0FAF	getallappsstatus
Fri Jan 04 15:04:21 PST 2008	OK	3.6	0014.4F01.0000.0FAF	hello
Fri Jan 04 15:03:07 PST 2008	OK	1.193	0014.4F01.0000.0FAF	getavailablesuites
Fri Jan 04 15:03:06 PST 2008	OK	1.181	0014.4F01.0000.0FAF	getallappsstatus
Fri Jan 04 15:03:03 PST 2008	OK	3.192	0014.4F01.0000.0FAF	hello
Fri Jan 04 15:01:19 PST 2008	OK	1.248	0014.4F01.0000.0FAF	getavailablesuites
Fri Jan 04 15:01:18 PST 2008	OK	1.233	0014.4F01.0000.0FAF	getallappsstatus
Fri Jan 04 15:01:14 PST 2008	OK	3.71	0014.4F01.0000.0FAF	hello
Fri Jan 04 14:40:23 PST 2008	OK	1.287	0014.4F01.0000.0FAF	getavailablesuites
Fri Jan 04 14:40:21 PST 2008	OK	1.196	0014.4F01.0000.0FAF	getallappsstatus
Fri Jan 04 14:40:12 PST 2008	OK	8.579	0014.4F01.0000.0FAF	hello

• System printout

Opens a window to display standard output from the main isolate operating on that SPOT. The image below shows standard output from the LightSensor application.



• Set Name

The first time a SPOT displays in Solarium, the second half of its full IEEE numerical address is used as its name and displayed at the top of the SPOT image. This command allows you to specify a more meaningful name as a replacement. The name will be stored in persistent memory on the SPOT itself.

• Enable OTA Commands

Allows the user to turn on the SPOT's OTA command server. This option is only enabled for USB-connected SPOTs that have their OTA command server turned off.

• Reset

Resets the SPOT. It has the same effect as pressing the Control button. It will restart the master isolate.

• Refresh

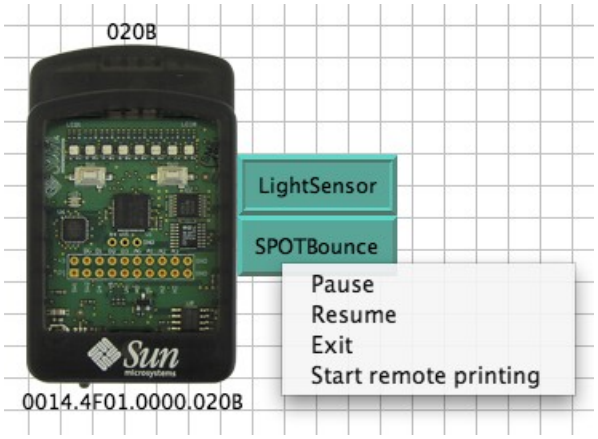
Queries the SPOT for its current list of installed and running applications and refreshes the display.

• Remove

Removes this SPOT from the Solarium display. Removed SPOTs can be added again through the SPOT discovery process.

Pausing, Resuming and Terminating Applications

Once an application has been launched in a child isolate, the user can pause, resume or terminate it. Clicking on the application object, brings up the menu shown below:



• Pause

Pauses execution of the selected application.

• Resume

Resumes execution of a paused application.

• Exit

Stops execution of the selected application.

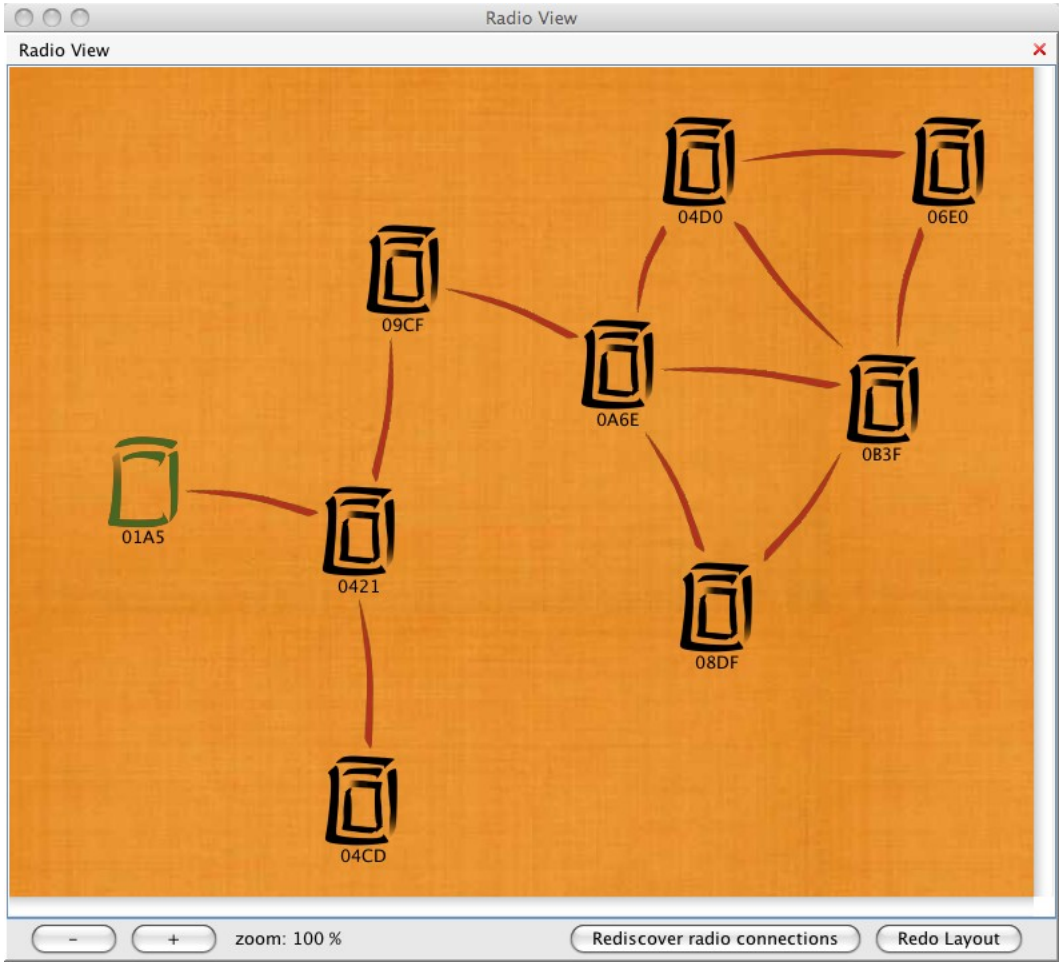
• Start remote printing

Opens a dialog box that directs System.out for the selected application to a window. The window can be a new window or a pane within the Solarium window.

Note: The current framework only supports pausing and resuming of simple applications, e.g., those that do not have currently active network connections.

Radio View

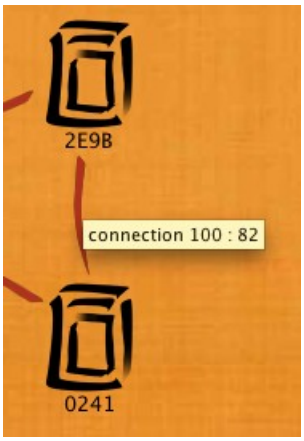
When debugging an application running on a network of SPOTs it can be quite helpful to be able to visualize the radio connectivity, i.e. which SPOTs can talk to each other. The *Radio View* provides a way for Solarium to discover and graphically depict the radio topology. Below is an example of a network of eight SPOTs and one basestation (shown in green).



When the Radio View window is first opened, Solarium sends commands to each known SPOT asking it to discover its neighbors. The red lines show which SPOTs that can exchange radio packets. After all the SPOTs have replied a graphing algorithm is used to create the best layout that shows the radio topology. The Radio View has no knowledge of where the SPOTs are located physically, so the layout algorithm can only try to minimize the number of connections that cross each other.

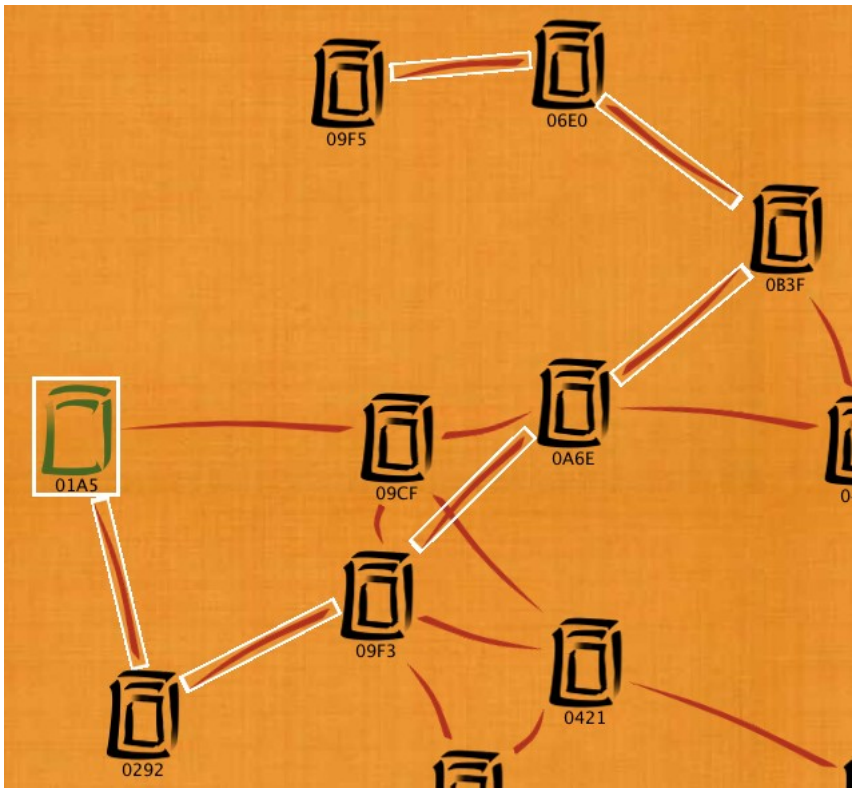
Since the radio topology can change over time, pushing the **Rediscover radio connections** button will cause Solarium to again ask each SPOT to discover its neighbors and then repeat the layout process. If the resulting layout is not clear enough, push the **Redo layout** button to try to find a better one.

If you hover the mouse over a radio connection a tooltip will appear showing the “Link Quality” in each direction as measured by the two SPOTs.



Link Quality is uniquely determined by the CORR value: CORR is a correlation measure provided directly by the radio part. Link Quality values range from 100% (the best, corresponding to a CORR value greater than 100) to 0% (the worst, which means the CORR value is less than 50).

The Radio View can also be used to show the route two SPOTs would use to talk to each other. Click on one of the SPOTs to bring up a pop-up menu where you can select the address of the SPOT to seek a route to. Below is the route from the basestation to 09F5.

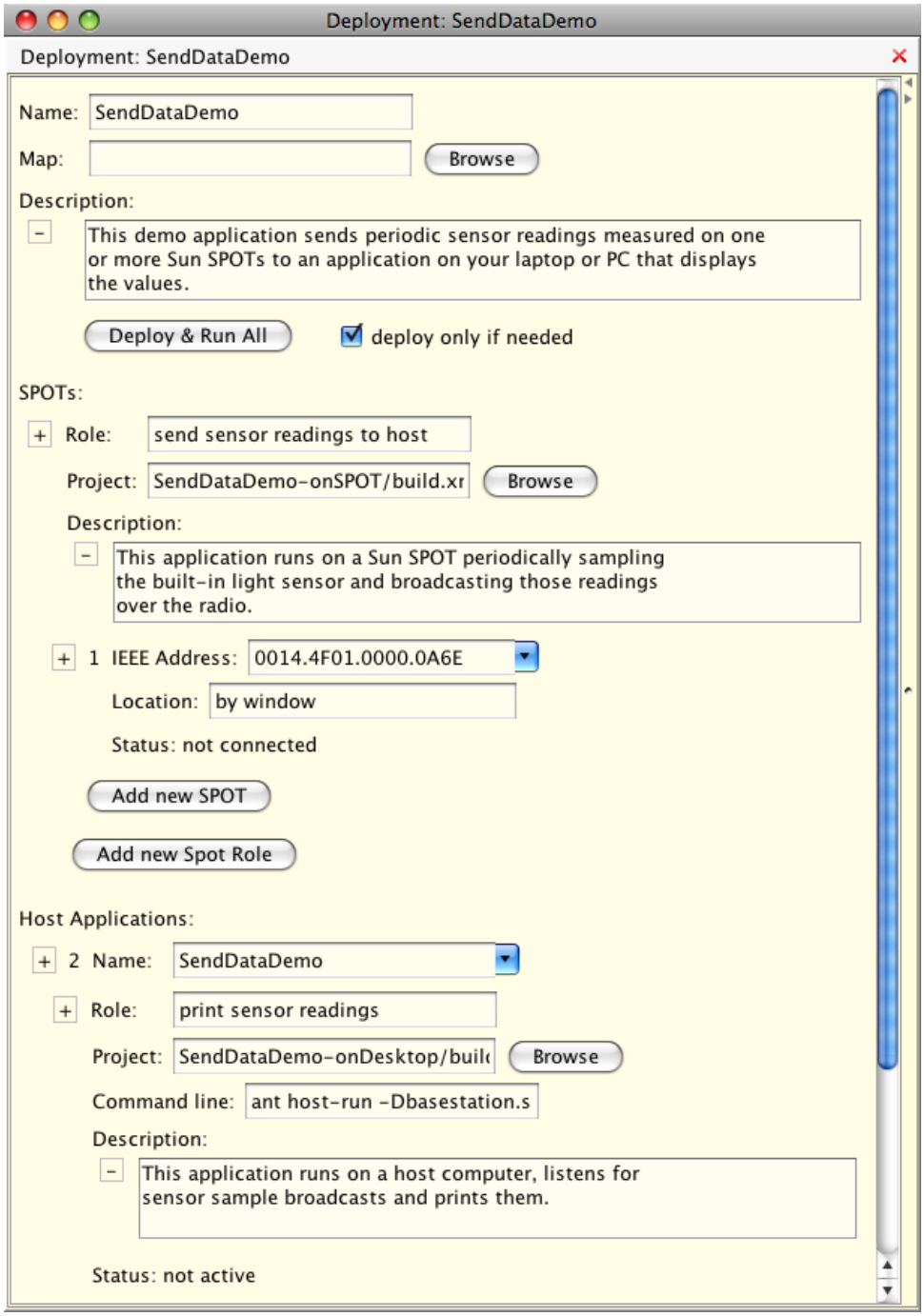


Note: SPOTs must have OTA enabled to participate in Radio View.

Managing a Network of SPOTs

Solarium provides a special *Deployment View* to make it easier to manage a network of Sun SPOTs. A deployment specifies what application should be loaded on each SPOT and allows you to deploy those applications with a single button press. The Deployment View also shows the current status of each SPOT. A deployment can also specify any associated host applications that need to be run.

Here is a simple deployment that describes the `SendDataDemo`.



The deployment information is in three main sections. First is some information about the deployment as a whole. This includes the deployment name, an optional image file to be used as a map of the deployment (not used above), and a text description of the deployment. Note the small box with a minus sign in it to the left of the descriptive text. Clicking it with the mouse will cause the text area to expand. Below the description is the **Deploy & Run All** button that can be used to load the latest version of your SPOT applications on all the SPOTs in the deployment and start them all up. Solarium tries to determine if the code currently installed on each SPOT is up to date or not. If the **deploy only if needed** checkbox is selected then any SPOTs that have the latest version of the application on them will not be updated.

The next section specifies what SPOTs are part of this deployment, what application they should run, and their current status. Since it is often the case that many SPOTs will be running the same application, this section consists of a series of *Roles*, one for each application used, where each role includes a list of SPOTs performing that role. Each role has a short one-line name, the filename for the `build.xml` file used to build the application, a longer description, and then a list of SPOTs to be loaded with this application. Each SPOT entry consists of the SPOT's IEEE radio address, an optional short description of where it is located, and its current status. At the end of the list of SPOTs is the **Add new SPOT** button used to add another SPOT to this role. Note that the SPOT's address has a pull-down menu that lists all SPOTs currently discovered by Solarium. This makes it easy to assign a SPOT to a role. The list of SPOTs also includes a **New Virtual SPOT** menu item if you wish to select an emulated SPOT. In the above sample deployment there is one role defined, with one SPOT associated with it.

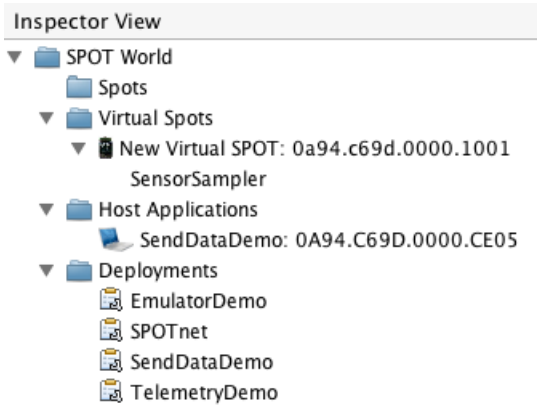
The final section specifies any host applications used by the deployment. The sample above has one host app that is used to receive the data broadcast by the SPOT and display it. Each host application has a name that the application will use to register itself when it starts up. When Solarium goes to discover SPOTs it now also will discover any host applications that have enabled OTA by calling:

```
OTACommandServer.start("SendDataDemo");
```

where `"SendDataDemo"` is the same as the host application name specified in the deployment. The remainder of the host application fields are similar to those for a SPOT Role with the addition of the command line to be used to start up the application. Normally this will just be `ant host-run`.

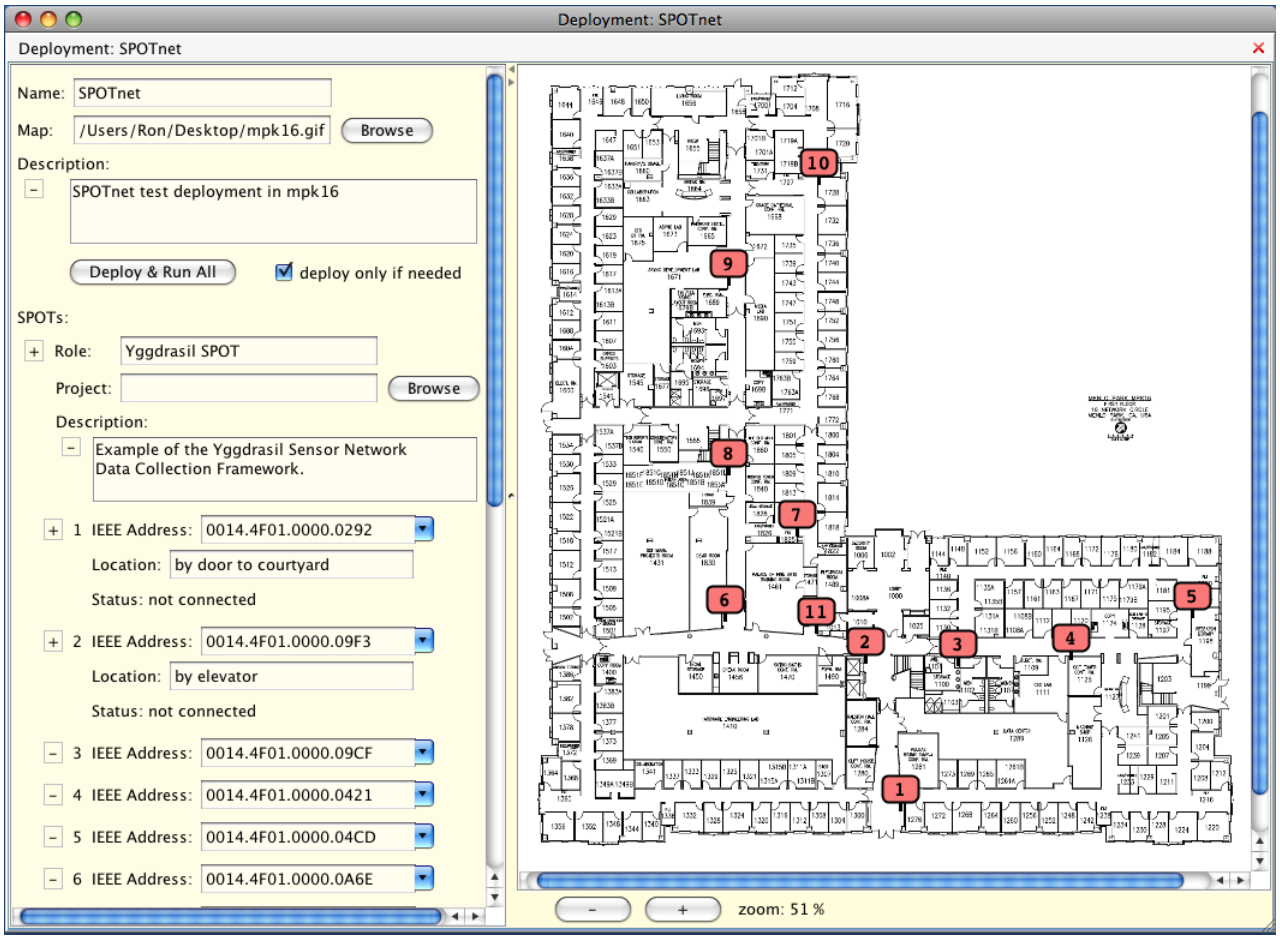
Note: Solarium must be started with the property `basestation.shared=true` in order to discover host applications.

Here is what the *Inspector View* looks like when using a virtual SPOT:



Note that the *Inspector View* lists all of the deployments that Solarium knows about. You can open one of them by right clicking on it with the mouse and selecting the **View** menu option. When you create a new deployment (**View → Deployment → New…**), a new file is created to hold the deployment information. The name you give the deployment is added to the list of deployments. The `SendDataDemo` included as part of the Sun SPOT SDK now also specifies a `deployment.xml` file that you can open (**View → Deployment → Open…**) that is the same as the sample deployment shown above.

Here is an example of a deployment of eleven SPOTs used at Sun Labs.



This deployment specifies a map file to use that is displayed in the right half of the window. Each SPOT is shown on the map with a small balloon containing its number. You can drag the balloon to wherever the SPOT is located. While not visible in the above screenshot, this deployment has a second role defined for SPOT number 11, which is running a different application than the other ten SPOTs.

Note: right clicking on a SPOT or its balloon will bring up a pop-up menu allowing you to deploy the application to the SPOT, run it, remove this SPOT from the deployment, and all of the normal Solarium commands for a SPOT.

Using the SPOT Emulator in Solarium

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.

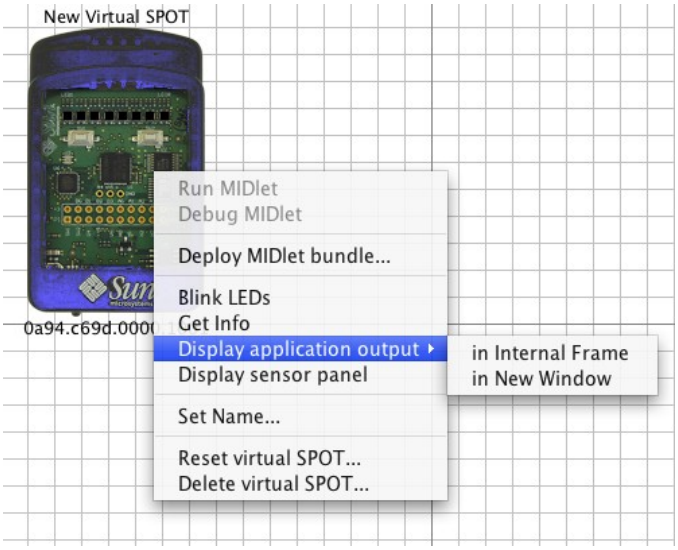
Instead of a physical sensorboard, virtual SPOTs have a sensor panel that can be used to set any of the potential sensor inputs (e.g. light level, temperature, digital pin inputs, analog input voltages, and accelerometer values). Your application can control the color of the LEDs that are displayed in 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.

Manipulating virtual SPOTs

Once Solarium is running make sure that a graphic *SPOT View* is displayed (**View** > **SPOT View**). Then from the **File** pull-down menu in the main Solarium menu bar, select the command **New virtual SPOT**. This will create and display a new virtual SPOT. It will appear to have a blue rather than a smoke-colored plastic case. You can use the mouse to place the virtual SPOT any place in the display.

If you right-click on the virtual SPOT you will see a menu of possible commands.



• Set Name…

This 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.

• Deploy MIDlet bundle…

This 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.

• Run MIDlet

Once you have loaded some MIDlets you can use this 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.

For example, 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.

• Debug MIDlet

This will list all of the MIDlets contained in the deployed jar file and allow you to connect an external Java Debugger to a MIDlet in order to debug it.

• Reset virtual SPOT…

This 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.

• Display application output

This command will 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 this command will bring it to the front.

• Get info

This 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.

• Delete virtual SPOT

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

From the **Emulator** pull-down menu in the main Solarium menu bar, 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...** pull-down menu allows you 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:

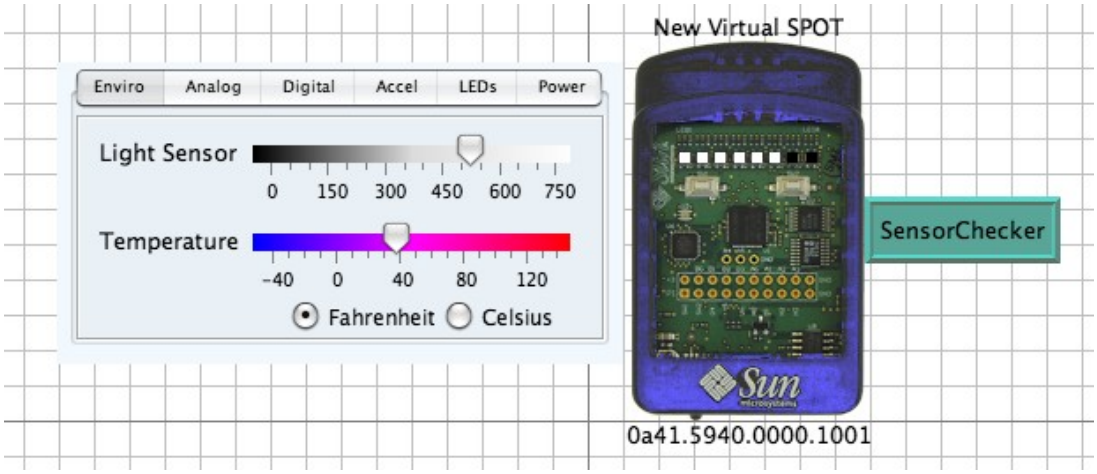
```
cd SunSPOT/sdk/Demos/EmulatorDemo
ant solarium -Dconfig.file=emulator_demo.xml
```

Using the Sensor Panel

Virtual SPOTs do not have a physical sensorboard so an alternate mechanism is needed to feed sensor input. Choosing the **Display sensor panel** menu option brings up a window with six tabs. Each tab corresponds to a set of input pins or sensors on the virtual SPOT and allows you to determine what values will be fed to the virtual SPOT. The tabs are:

- *Enviro* - Light sensor and temperature sensor
- *Analog In* - The input levels on analog pins A0 to A5
- *Digital Pins* - The input on digital pins D0 through D4. Also displays the output to pins H0 through H3.
- *Accel* - The input from the accelerometer
- *LEDs* - The input from the two switches on the sensor board. Also displays the output from the LEDs.
- *Power* - The input voltage from the battery and charging rate.

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 *Inspector View*. From the *Inspector View* the **Display sensor panel** command will create a new window to display the sensor panel. To locate a virtual SPOT in the *SPOT View* one can cause its LEDs to blink using the **Blink LEDs** command from its popup menu in the *Inspector View*.

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:

```
basestation.shared=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.

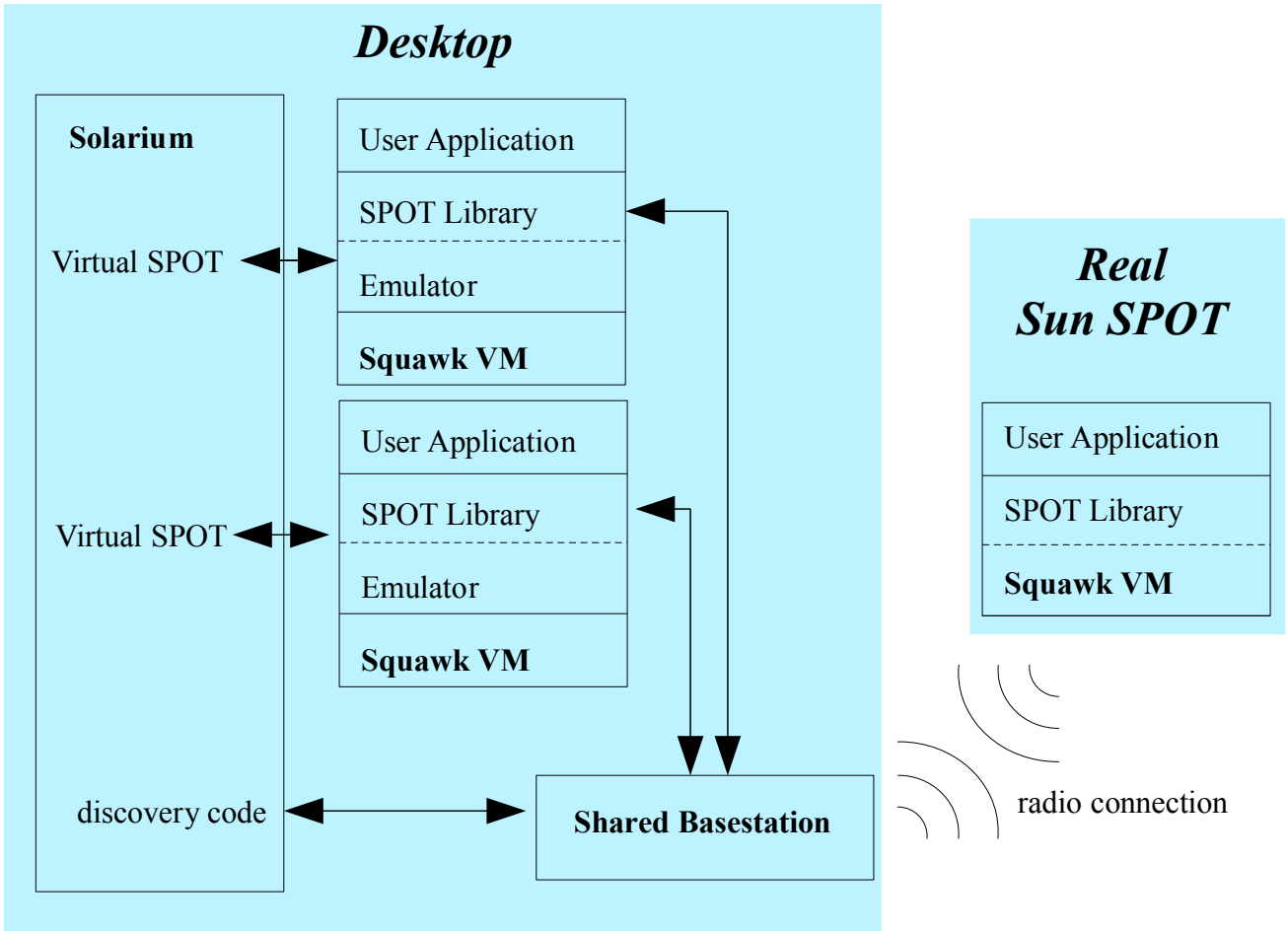
Note: virtual SPOTs can also communicate with SPOT host applications using the radio, but only if the host application is run with `basestation.shared` set to `true`.

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 it is worth explaining how the two terms are used here. 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 from the Emulator?

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.

Future Directions for the Emulator

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

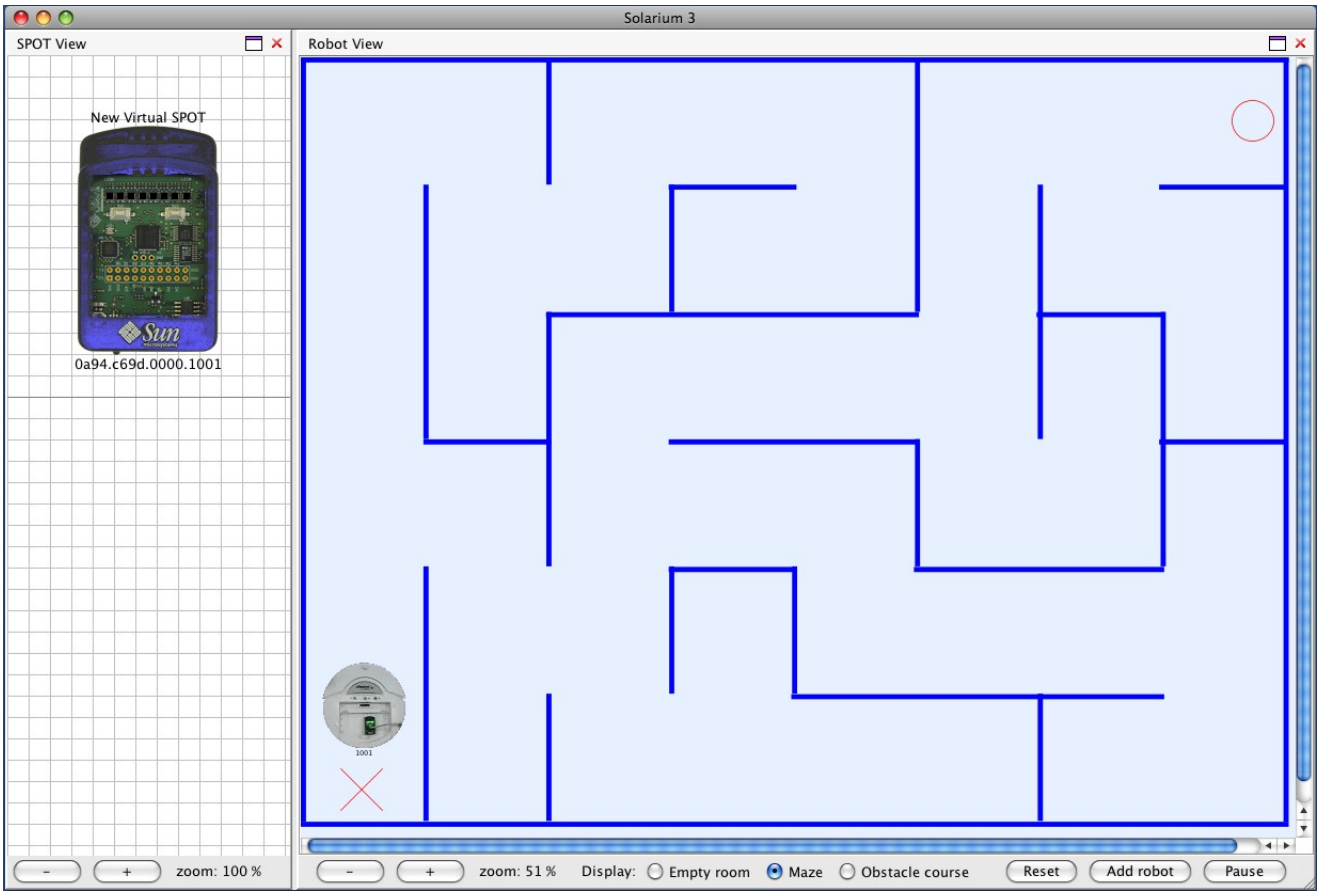
written Java code into Solarium and have it control the input to a virtual SPOT, e.g. to compute the acceleration on the SPOT as it is moved.

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.

Robot Emulator

As part of the Sun SPOT team collaboration with educators we have added to Solarium an emulator for the iRobot Create. The simulation roughly mimics the environment that is used in the International Autonomous Robotics Competition (iARoC) competition held in San Diego each year sponsored by Wintriss School. In this competition students use Sun SPOTs to control an iRobot Create to get it to navigate a maze. The Create is very much like its more popular sibling, the Roomba, except it can be easily controlled by an external computer, in this case a Sun SPOT.

In the simulation you write software for a Sun SPOT, and then in Solarium, create a **Robot View** from the **View** menu. Pushing the **Add robot** button will create a Robot/VIrtual Sun SPOT combination. Back on the SPOT View, you will now see a Virtual SPOT.



You deploy and run your software on this Virtual SPOT and it will control your simulated robot. You have your choice of three different environments to run in; an empty room, a maze, or an obstacle course. Each view includes an 'X' as a starting point and a 'O' as an ending point. Your robot includes sensors that allow you to sense when your robot is over one of these marks.

The iRobotCreateDemo in the SDK Demos folder provides a simple application that can use the Create's bump sensor to find the center of the room. Deploy the demo to the virtual SPOT and when you run it the Create will move under the SPOT's direction.

Note: the **Pause** button only affects the Create, it does not cause any application running on the SPOT to be paused. Similarly if the SPOT application quits without stopping the Create, e.g. it throws a fatal exception, then the Create will continue with whatever values were specified by the last drive command.

The Create API used by the Robot Emulator can also be used by a real SPOT to control an actual Create—all that is needed is to require the appropriate Create library jar file in your project's `build.properties` file.