Sun™ SPOT Owner's Manual

Red Release 5.0

Sun Labs
June 2009

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300
Part No. 820-1249-10
Document Revision 1.9
June 2009

Contents

Sun™ SPOT Owner's Manual

This document provides a quick introduction to the Sun SPOT (Small Programmable Object Technology) kit, for Red software release 5.0.

Contents of the Sun SPOT Kit

A Sun SPOT kit contains the following:

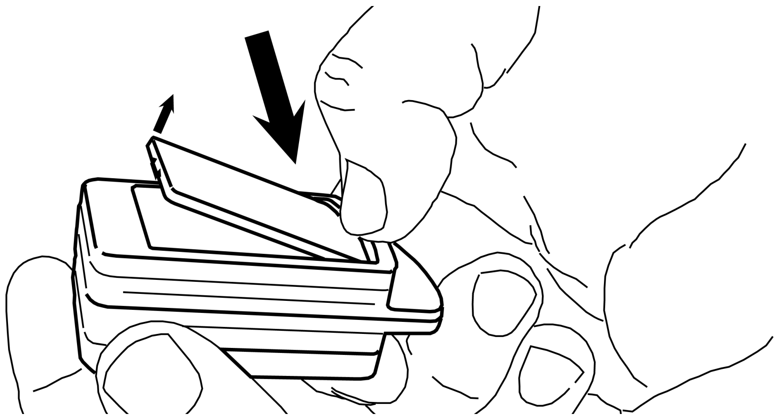
- one basestation Sun SPOT unit with USB power
- two free-range Sun SPOT units with onboard battery
- a USB cable for connection between a standard USB port and a Sun SPOT unit
- one Sun SPOT software CDROM
- two mounting brackets, each allowing a Sun SPOT unit to be wall-mounted
- one mounting bracket to allow mounting of a Sun SPOT to a circuit board

Dimensions of the Sun SPOT are 41 x 23 x 70 mm. Weight is about 54 grams.

A Sun SPOT can enter deep sleep mode, when it consumes very little power.

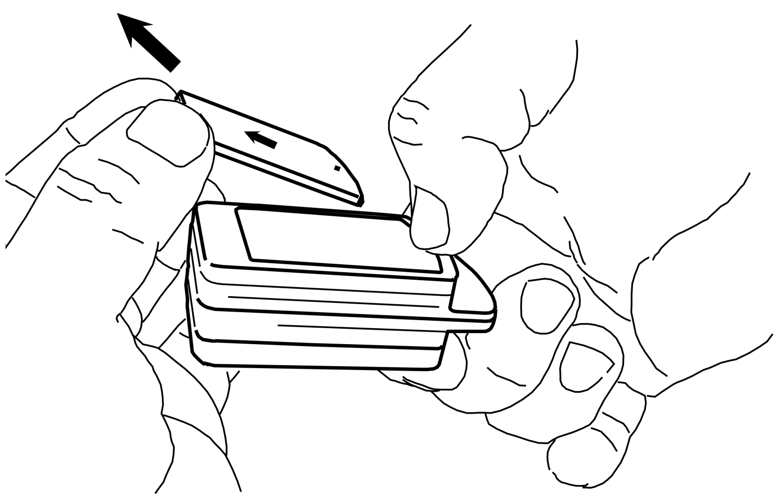
How to Open a SPOT

You must open the Sun SPOT unit lid to be able to reach the switches and LEDs on the sensor board. To open the lid, press down *firmly*, down and back, on the edge of the lid near the small raised dot. You can think of that small raised dot as the fingernail-catching dot. The closer to the edge of the lid that you press, the easier the lid will open. The opposite end of the lid pops up.



Press down firmly on the edge of the lid marked with a small raised dot.

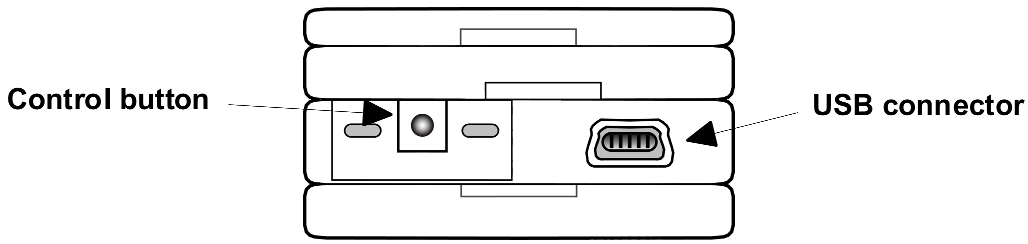
After the lid pops up, pull the lid out and away.



Pull the lid out and away.

Guided Tour of SPOT Switches and LEDs

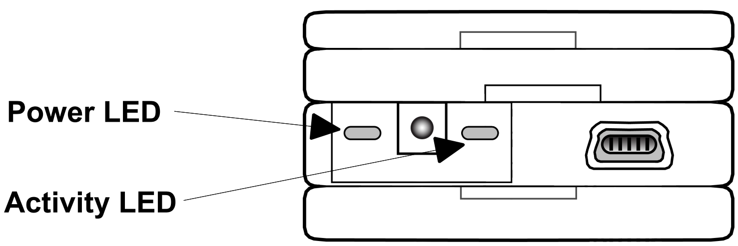
The Sun SPOT unit has one switch and one connector that are accessible without removing the case lid. These are shown below:



The connector is the micro-USB connector that allows the Sun SPOT unit to be connected to a host workstation.

The switch is the Sun SPOT unit control switch. If the Sun SPOT unit is off, pressing the switch turns the Sun SPOT unit on and causes it to boot. If the Sun SPOT unit is on, pressing the control switch causes the Sun SPOT unit to reboot. If the Sun SPOT unit is on, pressing the control switch and holding it down turns the Sun SPOT unit off.

This end of the Sun SPOT unit also has two LEDs behind the plastic casing.



The power LED is to the left of the power switch. This LED exhibits the following behaviors.

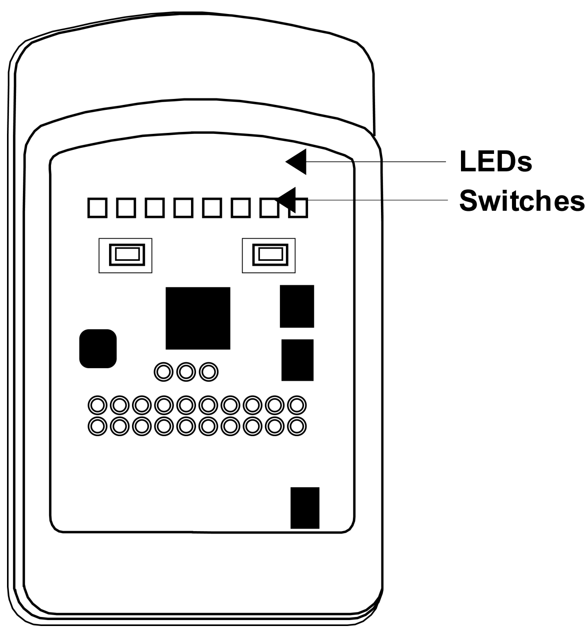
Power State	**Power LED Behavior**
Powering up	One bright green pulse, sharp on, soft off
Powering down	Three bright red flashes
Charging the battery while CPU is active	Slowly alternate between a dim green and a bright green on a eight second cycle
Charging the battery when CPU is asleep	Slowly alternate between off and a dim green on an eight second cycle
External power supplied, but not charging, CPU active	Steady dim green
Battery low	Steady dim red. *This is a change from Release 1.0.*
Power fault	Two short red flashes

Power State	Power LED Behavior
CPU going to sleep	Short red flash, short green flash
External interrupt or alarm, including button tap.	One short green flash

The activity LED is to the right of the power switch. This LED is under Java program control and can be used in your applications, but it is usually used by the system software. Some of these uses are:

- When the Sun SPOT unit is attempting to synchronize with a host workstation, the activity LED flashes amber 16 times a second. This flashing lasts for two seconds or until synchronization is complete, whichever comes first.
- When the Sun SPOT unit is being used as a basestation, that is, for wireless communication between a host workstation and free-range Sun SPOT units, the green component of the activity LED changes state, i.e., switches from off to on or the reverse, for every packet received on the Sun SPOT unit from the host workstation. The red LED component of the activity LED changes state for every packet sent to the host workstation from the Sun SPOT unit. If no packets are being sent or received, then the green activity LED blinks twice every 12 seconds to indicate that the basestation is functioning.

If the Sun SPOT unit lid has been removed, there are two switches and eight multi-color LEDs that become accessible.



The LEDs have red, blue, and green components. The switches and LEDs have no fixed purpose and are under Java program control.

Getting Started with Sun SPOTs

The Sun SPOT SDK comes with a brief tutorial that teaches you the basics of using and programming the Sun SPOTs. We strongly suggest that you run through it before you do anything else with your SPOTs. It takes half an hour or less and is well worth the time. It is the best way to get started with your Sun SPOTs.

The tutorial is a series of web pages, with the first page located at

```
[SpotSDKdirectory]/doc/Tutorial/Tutorial.html
```

where `[SpotSDKdirectory]` represents the directory in which the SPOT SDK was installed. On a Windows machine, this would typically be:

```
C:\Program Files\Sun\SunSPOT\sdk
```

SPOTManager Tool

The Sun SPOT SDK comes with two important tools for managing the software on your SPOTs: SPOTManager and Solarium.

The SPOTManager is a tool for managing the Sun SPOT SDK software. You can use it to download from the Internet both new and old versions of the Sun SPOT SDK. You can use it to make one or another SDK the active SDK on your host workstation, and you can use it to download system software to your Sun SPOTs.

Starting SPOTManager

The SPOTManager tool is implemented as a Java Network Launchable Program (JNLP) file. To start the SPOTManager tool, connect to:

```
http://www.sunspotworld.com/SPOTManager/
```

and click the image of a SPOT, or, alternately, access the URL:

```
http://www.sunspotworld.com/SPOTManager/SPOTManager.jnlp
```

and the latest version of the SPOTManager application downloads to your host workstation and starts.

If you start SPOTManager with one of these URLs, you always have the latest copy of the SPOTManager software. However, if you want to keep a cached copy of SPOTManager to operate when your host workstation is not connected to the Internet, follow these steps:

1. **Launch the Java Control Panel**

   In Windows, this is under *Start* > *Control Panel*. The Java control panel may not be visible if your control panels are organized in category view. If this is the case, select *Switch to Classic View* to gain access to the Java control panel.

2. **Under the General tab, go to the Temporary Internet Files panel and click the *View* button.**

This should display the Java Cache Viewer.

3. **Select the SPOT Manager application. Click the shortcut icon (an arrow pointing up and to the right) in the menu bar.**

   This should create a local shortcut to the SPOTManager application as stored in the cache.

Using the SPOTManager Tool

The SPOTManager tool, as currently configured, has six tabs across the top:

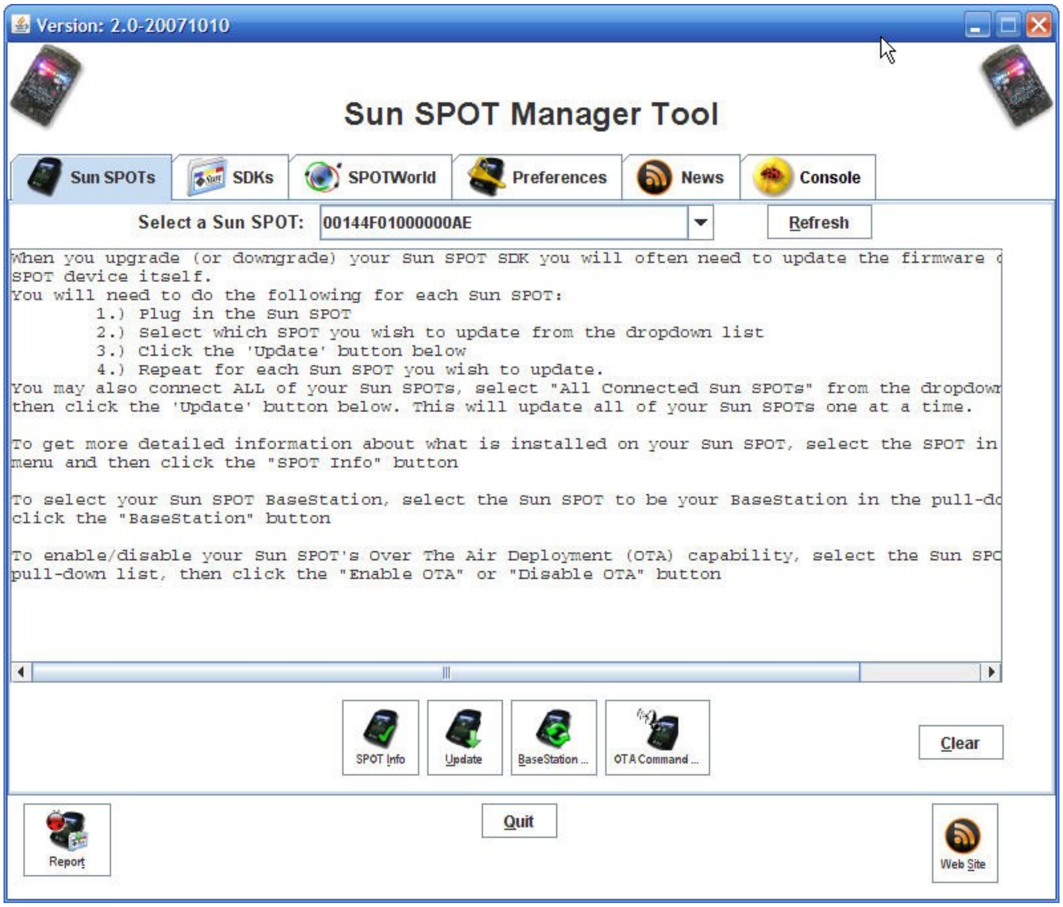
- Sun SPOTs – allows you to query the configuration of individual Sun SPOTs, download and configure the system software on them.
- SDKs – allows you to download and install versions of the Sun SPOT SDK available on the Sun SPOT website.
- Solarium – launches a tool for managing individual Sun SPOTs and the application software on them. This tool also contains a Sun SPOT emulator capable of running and testing Sun SPOT application software.
- Preferences – configures the network and polling characteristics of the SPOT Manager tool itself.
- News – gives you access to the contents of discussion groups on the Sun SPOT website.
- Console – displays standard output for the SPOTManager tools. If there is new error output here, the type for the console tab label is red. Otherwise, it is black.

It also has three buttons across the bottom of the window, no matter which tab is displayed. The buttons are:

- Report – allows you to report a bug in Sun SPOTs, Sun SPOT software, or the SPOTManager. If you allow, it collects and includes configuration information on the connected Sun SPOTs at the time the report is produced.
- Quit – quits the SPOTManager tool.
- Website – opens a web browser with the URL set to the Sun SPOT website, `www.sunspotworld.com`.

Sun SPOTs Tab

The Sun SPOTS tab has a menu for selecting among the USB-connected Sun SPOTs, a large text output area, and five buttons particular to this tab.



The menu above the console area is labeled "Select a Sun SPOT" and the pull-down menu lists all the Sun SPOTs that are connected by USB cables to the host workstation. Each Sun SPOT is listed by its IEEE network number. The first eight digits of this number are always 0014.4F01. The last eight digits are written on a sticker visible through the plastic surrounding the radio antenna fin on the Sun SPOT.

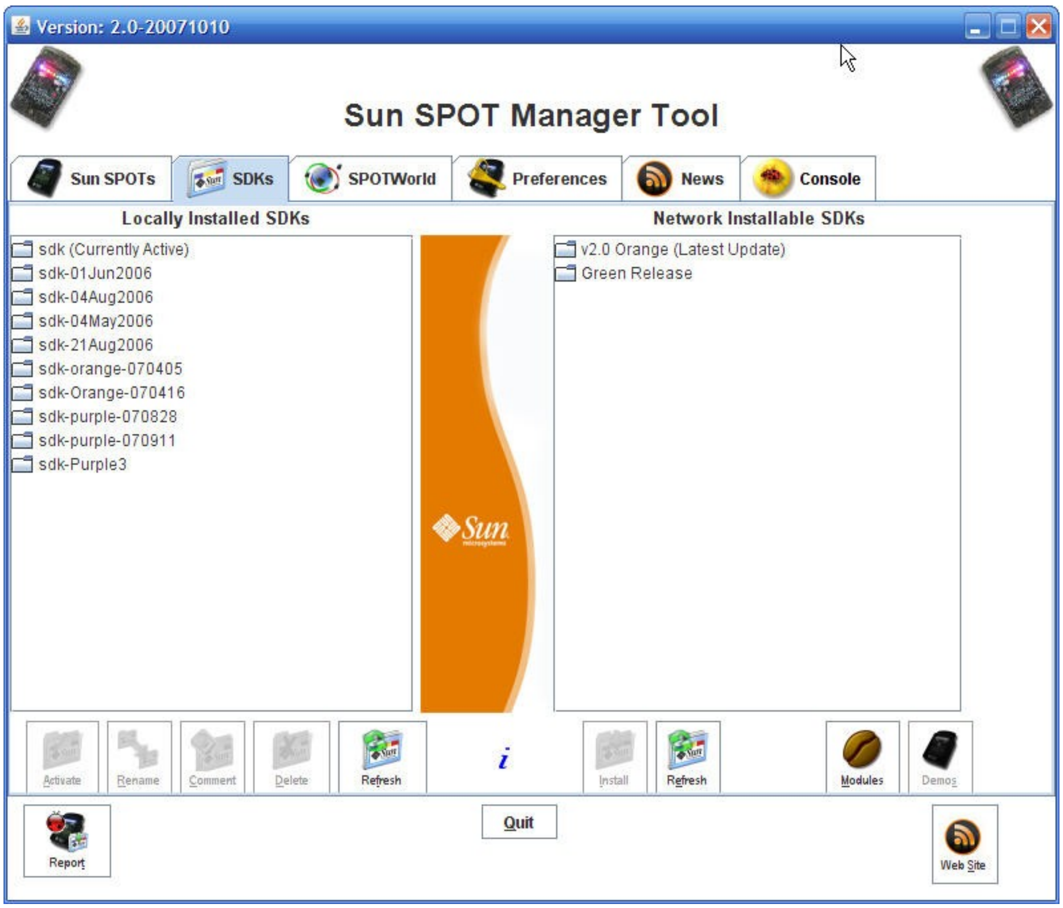
The five buttons all take an action with respect to the selected Sun SPOT. The five buttons are described below.

Button	Description
SPOT Info	Runs `ant info` on the selected Sun SPOT and displays the information in the output area. The `ant info` command output describes the software configuration of the SPOT.
Update	Does an `ant upgrade` on the selected Sun SPOT, loading the SPOT with the firmware required for the current version of the SDK.
Basestation	Brings up a pop-up menu where you can enable or disable the selected SPOT as a basestation. This is ordinarily done on a Sun SPOT that does not have a battery or a demo sensor board. You are given a choice of a plain basestation (more efficient, can only talk to one free-range SPOT at a time) or a shared basestation (more overhead, can talk to multiple free-range SPOTs).

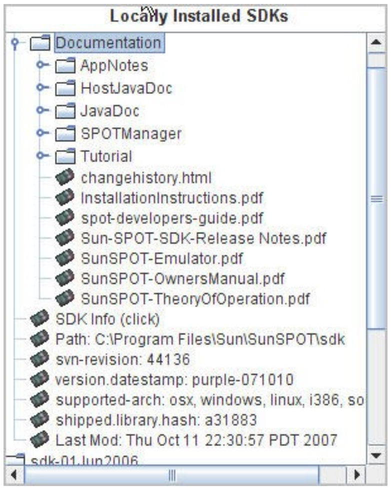
Button	Description
OTA Command	Enables or disables Over The Air (OTA) command processing on the selected SPOT. A pop-up menu allows you to select either Enable OTA of Disable OTA. Enabling OTA allows you to deploy, run, and debug application programs on a Sun SPOT using the radio link between the basestation and the selected Sun SPOT. **Note:** The OTA command server should only be enabled on the free-range SPOTs. It is not needed nor intended to be enabled on the basestation SPOT.
Clear	Clears the output window of all text.

SPOTManager SDKs Tab

The SDKs tab shows the Sun SPOT SDKs that are available on the local workstation. It also shows the SDK versions that are currently available on the Sun SPOT website.



The left column lists the Sun SPOT SDK versions installed on the host workstation. The right column lists the Sun SPOT SDK versions available on the Sun SPOT web site. Clicking the folder icon for an SDK expands the information displayed and, in some cases, provides clickable links to documentation available as part of that SDK.



The buttons below the lists of SDKs take action on or with the SDKs. The buttons in the Locally Installed SDK menu are described below.

Button	Description
Activate	Makes the selected locally installed SDK into the active SDK; records this SDK in the user's `.sunspots.properties` file as the SDK to be used for all `ant` commands.
Rename	Allows you to rename the selected Sun SPOT SDK directory. **Note:** Under Windows it is not always possible to rename the active SDK.
Comment	Allows you to add a comment line to the SDK. This is displayed when the SDK node is expanded as shown in the figure above this table.
Delete	Deletes the selected SDK from the local disk.
Refresh	Refreshes the list of local SDKs.

To the right of these buttons is a script "i" icon. Clicking this icon opens a window displaying a help file for the SPOTManager.

There are two buttons immediately to the right of the "i" icon and immediately under the Network Installable SDKs menu. They are described below.

Button	Description
Install	Downloads the SDK from the Sun SPOT website and installs it on the host workstation. Clicking this button first opens a software license window. After you accept the license terms, it opens a pop-up window where you can specify the directory where the SDK will be installed. If the SDK name is already in use, the system attempts to rename the previous directory. If it cannot rename that directory, the install fails.
Refresh	Polls the Sun SPOT website to see which Sun SPOT SDKs are available.

To the right of the network install and network refresh buttons are two more buttons and one icon:

Button	Description
Modules	If the coffee bean is a tan color and the “Modules” label under it is dark, you have enabled NetBeans modules in the NetBeans development tool and there is a new Sun SPOT module available on the website for download. Clicking this button downloads and installs the Sun SPOT module. If the coffee bean is a dark roast color and the “Modules” label is gray, then you have either downloaded the most recent module or modules are not enabled in your configuration of NetBeans. The button is inactive.
Demos	If the LEDs on the Sun SPOT in the icon are lit and the “Demos” label is black, then new demo programs are available on the Sun SPOT website. In this case, clicking the button launches a dialog box for downloading the latest demos. If the LEDs on the Sun SPOT icon are not lit and the “Demos” label is gray, then you have the latest demos in your SDK and the button is inactive.

Solarium Tab

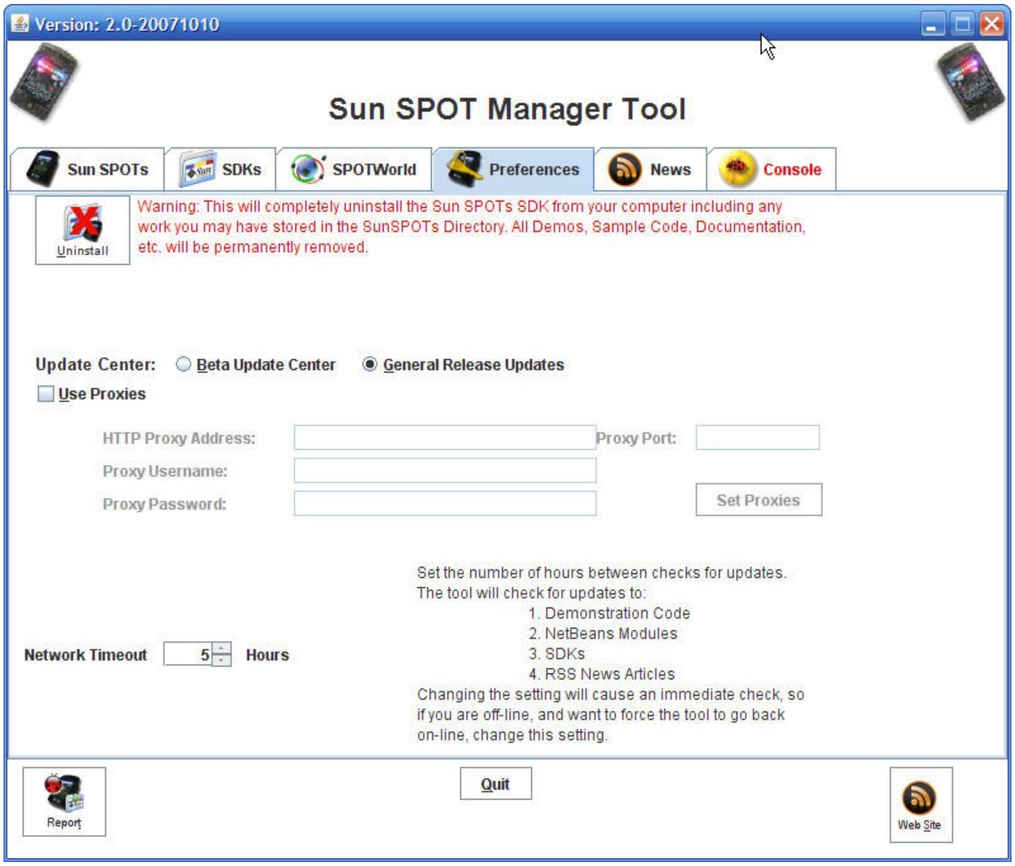
This tab allows you to start an application called Solarium. Solarium makes it easier to manage a group of Sun SPOTs and the application software for those SPOTs. Solarium also contains a SPOT emulator that is useful for testing SPOT software. The use of Solarium is explained further in the section “Solarium Tool” on page 18.

Solarium enables a basestation SPOT to communicate with any real Sun SPOTs. A basestation is not required if you only want to communicate with emulated SPOTs. A shared basestation is required if you want to have emulated SPOTs communicate over the air with real SPOTs.

If you start Solarium by using the Solarium button here, SPOTManager displays a dialog box asking you to specify which of the USB-connected SPOTs is intended to be the basestation for Solarium.

Preferences Tab

The Preferences tab allows you to control the operation of the SPOTManager tool itself.



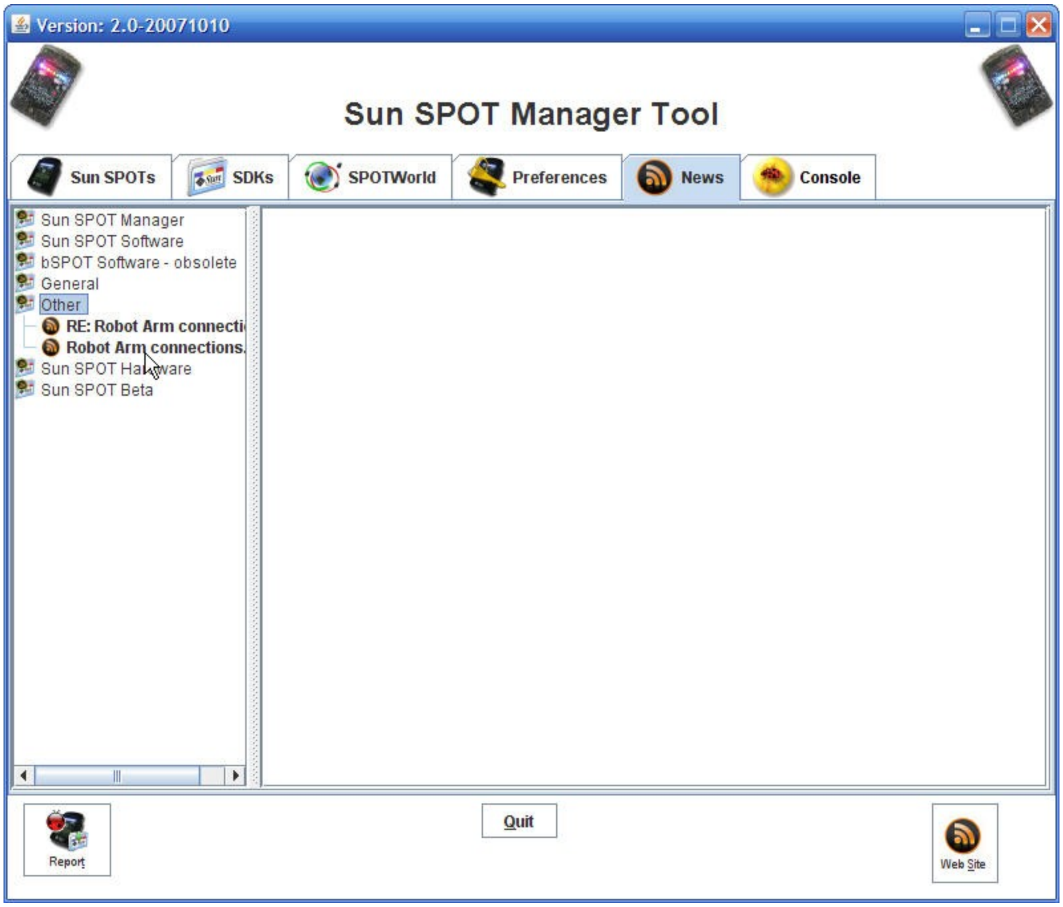
Selecting the *Uninstall* button removes all Sun SPOT SDK directories from your host workstation, including the demos and the documentation.

The Update Center settings allow you to specify which SDKs are listed as downloadable on the SDK page. If “Beta Update Center” is selected, then fully released software and beta software are shown. If “General Release Updates” is selected, then only fully released software appears.

If your Internet connection requires you to use proxies, select “Use Proxies” and specify the proxy host address, port, username, and password.

Finally, the “Network Timeout” setting controls how often the SPOTManager tool checks the Sun SPOT server for updates. Changing this setting also causes the software to refresh the listings immediately after the change is made.

News Tab

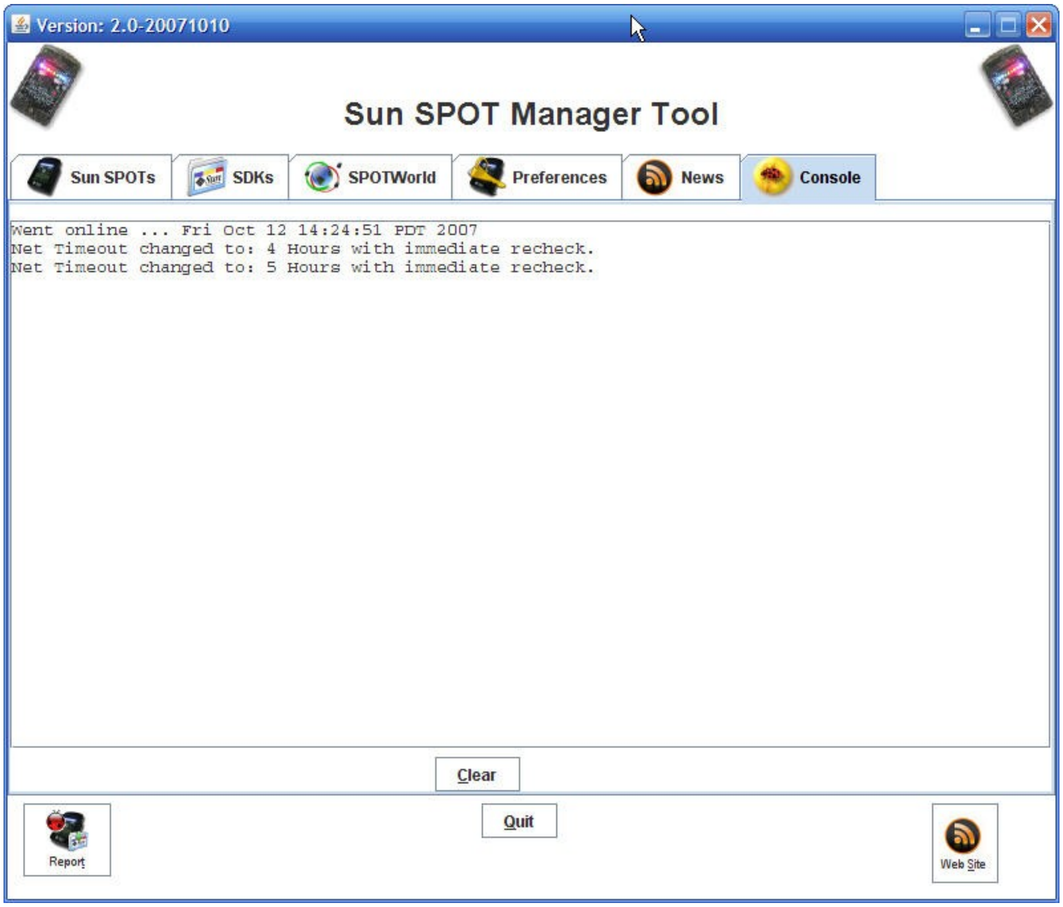


The News tab gives you access to the postings on the discussion pages on `www.sunspotworld.com`. The left side lists the topics. Opening one of the labels lists the subjects of the postings made under that topic. Clicking an individual posting causes it to be displayed in the large panel to the right.

Console Tab

The Console tab displays `System.out` and `System.err` for the process in which SPOTManager itself is running. Also, the output for any daughter processes that has not been redirected to its own output console goes to this display. For example, if you issue an Upgrade command in the Sun SPOTs tab, because the Sun SPOTs tab has an output area, the output goes to that output area. However, if you were to change the Network Timeout value under the Preferences tab, this tab does not have an output window of its own, so the output from that command goes to the Console display here.

The *Clear* button empties the console display.



If you have a problem with a command, useful error messages are often displayed here.

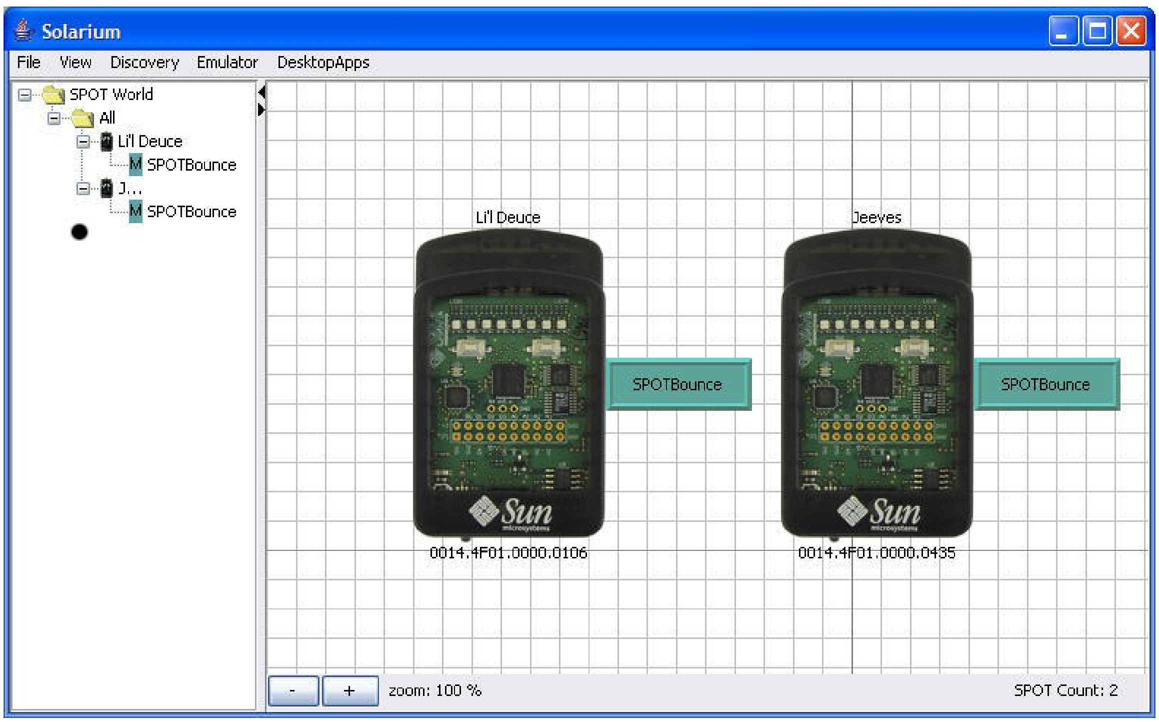
Solarium Tool

Solarium is a host workstation application that manages application software on Sun SPOTs.

Solarium can be started from the Solarium tab in SPOTManager by clicking the Solarium button. You can also start Solarium by opening a command line window, navigating to any SPOT project directory (containing `build.xml` file) and executing the command `ant solarium`.

The Solarium application may take a few minutes to find all of the SPOTs that are within radio range and running the OTA command server. To learn how to start the OTA command server on a SPOT, see “Sun SPOTs Tab” on page 11.

After Solarium finds the SPOTs, the tool displays them and the applications they are running, as shown below.



You can display SPOTs in a tree-like Inspector view, shown here to the left, or a 2-dimensional SPOT view, shown here against the quadrille background to the right. The views are controlled through the View pull-down menu on the menu bar.

The first time a SPOT displays in Solarium, the IEEE numerical address is used for the SPOT’s name.

Isolates and Jar Files

Before we discuss the things that can be done with SPOTs in Solarium, we need to introduce the topics of applications, MIDlets, isolates, and `jar` files.

In Java SE, an application consists of a static `main()` method defined in one of the loaded classes or it extends the Applet class if it is to run in a browser. In Java ME, an application is defined as a class that extends the *MIDlet* class. The Squawk VM used by Sun SPOTs implements Java ME, so all Sun SPOT applications extend the MIDlet class.

In standard Java ME, only one application can be run at a time in a Java VM, though that application may consist of many threads. Squawk allows for multiple applications to be run together in a single SPOT and uses a special *Isolate* class to prevent one application from interfering with the execution of another. Each MIDlet-based application is run in a separate isolate. While one isolate cannot directly access the instances in another, they all share the same underlying SPOT resources.

Some resources are unique, such as a radio connection on a specific port number. The first isolate to ask for that port is successfully given access to it, while any subsequent requests from other isolates fail. Other resources are truly shared such as the LEDs: one isolate might turn an LED on, and then another might turn it off or change its color.

The isolate controlling the Sun SPOT at the system level is the *master isolate*. There is only one master isolate per SPOT. The master isolate controls the radio stack for the SPOT and access to the other components of the SPOT. Normally a SPOT application consists of a single MIDlet that is run in the master isolate.

Solarium allows you to start and stop multiple MIDlets on a SPOT. These MIDlets are run as *child isolates*. Child isolates may suffer a slight performance penalty; for example, their radio transmissions have additional overhead because the child isolate must send requests to the master isolate to access the radio.

Java ME allows you to package up several MIDlets into a single *jar file*. The MIDlets must all be listed in a special file, `manifest.mf`. The `manifest.mf` file can be found in your project's `resources/META-INF` directory.

When you add a new MIDlet to your project you must add a new line for it in the manifest file. If you use the SPOT modules for NetBeans, you can use the pop-up menu on the package (select the *New > File/Folder* command and then select *MIDlet Class* from "Java Classes") to create the new MIDlet. NetBeans automatically updates the manifest file. NetBeans also updates the manifest if you rename an existing MIDlet.

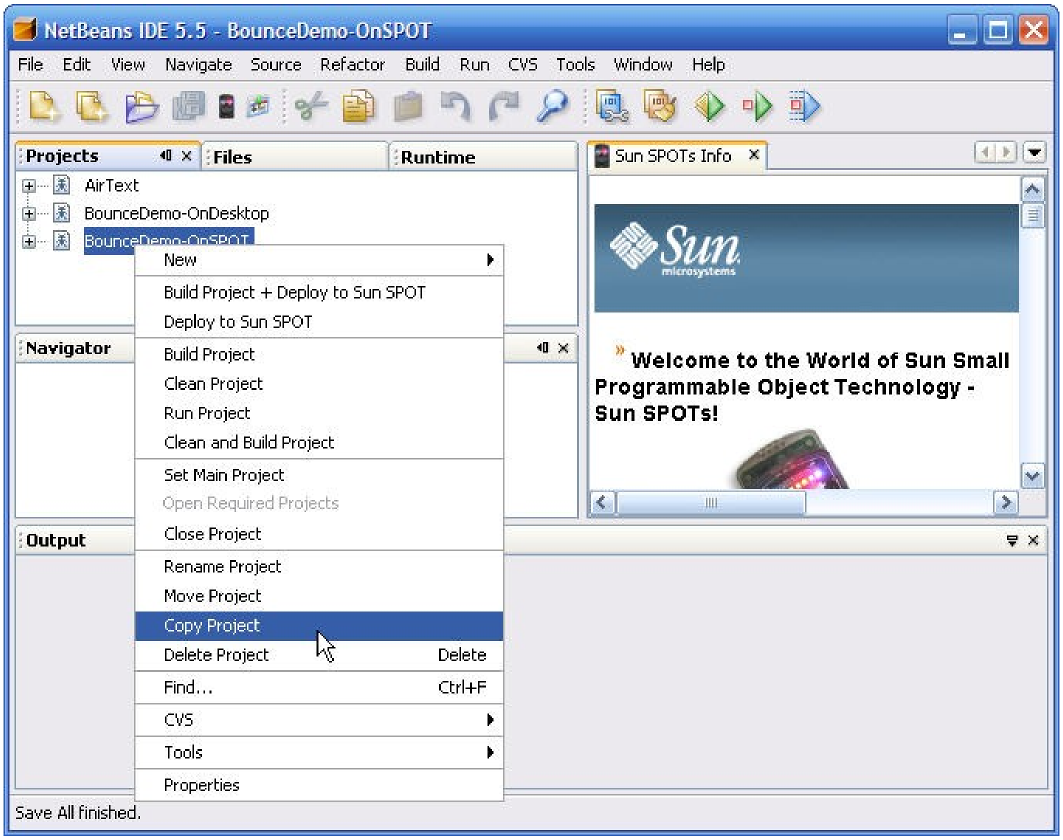
When the `jar` file is deployed to a Sun SPOT, all of the MIDlets are then available to be started from Solarium. One of them may be specified to be automatically run in the master isolate when the SPOT is rebooted. For now only one `jar` file may be deployed on a SPOT, but in the future it may be possible to deploy several.

Example: Creating a Jar File with Multiple MIDlets

With this theoretical background in place, we will briefly go through an example of the creating of a jar file with two MIDlets. We will use two existing demonstration projects: the Bounce demo and the AirText demo. If you have worked your way through the Sun SPOT tutorial, as we strongly suggest, you will be familiar with these demos.

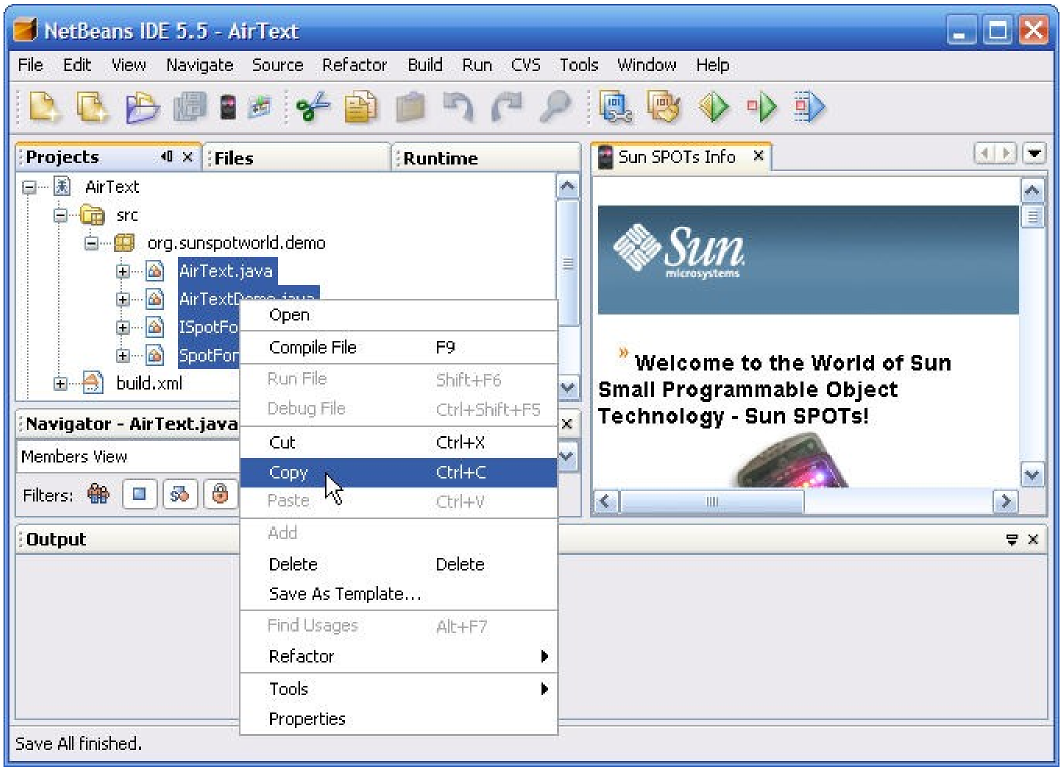
Briefly, we will copy the classes from the Bounce demo and the AirText demo into a new project, then edit the manifest file. After that, we will compile the new jar file and it will be available to deploy to a Sun SPOT.

1. **Open NetBeans. Select the *Project* tab in the upper left, and select the "BounceDemo-OnSPOT" project. Right click the project and select *Copy Project*.**



A dialog box displays, allowing you to specify the new project name. Here, we give it the name `ExampleJar`. If we were to compile this project now, we would get a `jar` file with one MIDlet in it, the Bounce demo. Now we need to add the AirText demo to the same project.

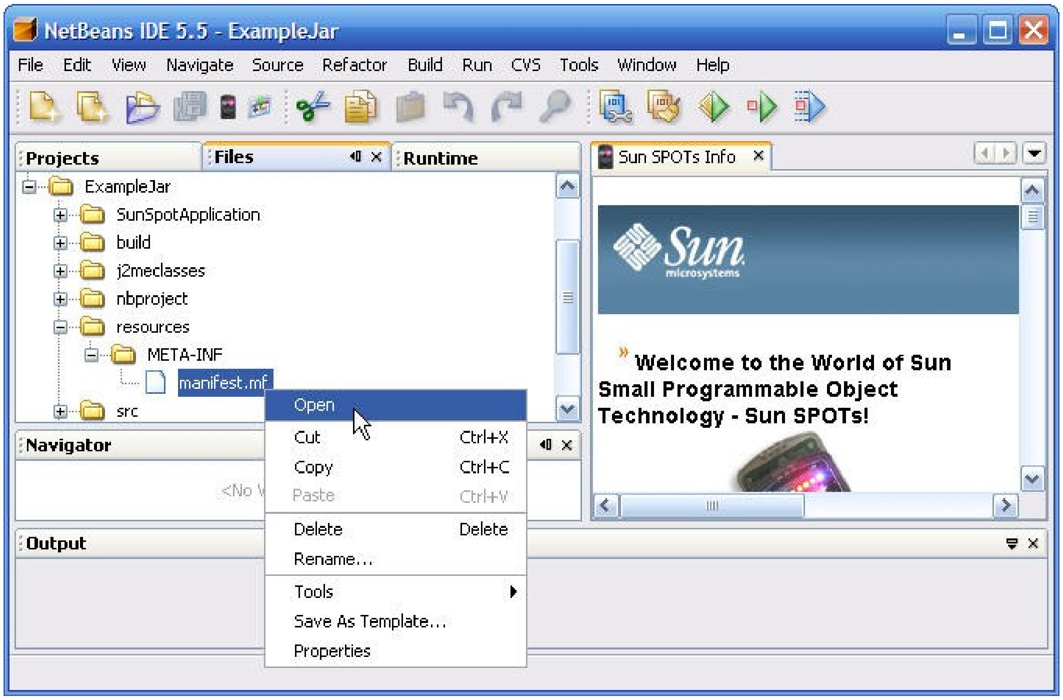
2. **Select the AirText demo, click the "+" to its left, then open up the `src` selection with in it, then open the `org.sunspotworld.demo` selection within `src`. Select all four Java classes, right click, and select *Copy*.**



3. **Select the `ExampleJar` project, click the "+" to the left of it, then open up the `src` selection with in it, select the `org.sunspotworld.demo` node within `src`, right click `org.sunspotworld.demo`, and select *Paste*.**

   The AirText classes should now be copied into the `ExampleJar` project.

4. **Select the Files tab in the upper left of NetBeans. Open the ExampleJar project, and the resources node within it. Open the resource META-INF and open the manifest.mf file with it.**



The manifest file displays in a new window to the right:

```
MIDlet-Name: eSPOT Bounce Demo-OnSPOT
MIDlet-Version: 1.0.0
MIDlet-Vendor: Sun Microsystems Inc
MIDlet-1: ,, org.sunspotworld.demo.SPOTBounce
MicroEdition-Profile: IMP-1.0
MicroEdition-Configuration: CLDC-1.1
```

The format of the manifest file is
<property-name>: <space><property-value>

The individual MIDlets are specified with property names of “MIDlet-1,” “MIDlet-2,” and so on. The MIDlet property value is a string of three comma-separated, arguments. The first argument is a name for the application, the second is intended to define an icon, but is not used in Sun SPOTs, and the third specifies the application’s main class.

We will edit the manifest file to give it a new MIDlet name and tell it where to find the AirText demo. We will also give the MIDlet descriptions some user-friendly names.

5. **Replace the MIDlet name with “Example Jar with two MIDlets.” Edit the MIDlet-1 line to include the phrase “Bounce Demo” between the colon and the first comma. Add a second line, below it, that reads:**

```
MIDlet-2: Air Text Demo, ,org.sunspotworld.demo.AirTextDemo
```

The manifest file should now read:

```
MIDlet-Name: Example Jar with two MIDlets
MIDlet-Version: 1.0.0
MIDlet-Vendor: Sun Microsystems Inc
MIDlet-1: Bounce Demo,, org.sunspotworld.demo.SPOTBounce
MIDlet-2: Air Text Demo, ,org.sunspotworld.demo.AirTextDemo
```

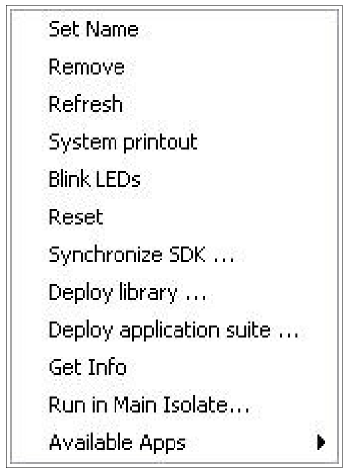
```
MicroEdition-Profile: IMP-1.0
MicroEdition-Configuration: CLDC-1.1
```

6. Save the new version of the manifest file.

If you were to compile the project now, you would have a single jar with two MIDlets in it. We can use that jar file in Solarium.

Manipulating Sun SPOTs in Solarium

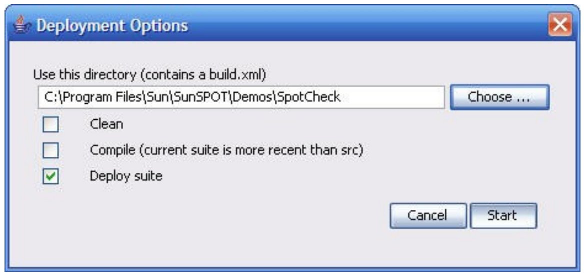
If you right-click a SPOT, the following menu displays:



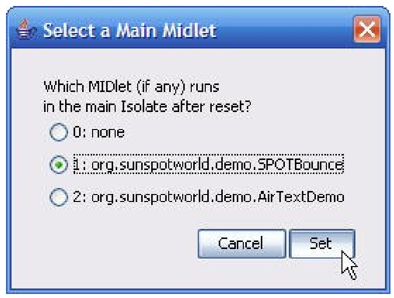
The choices in this menu are:

- Set Name – Allows you to specify a name to replace the IEEE number in Solarium displays. This name is stored in persistent memory on the SPOT itself.
- Remove – Removes this SPOT from the Solarium display. Removed SPOTs can be added again through the Discovery menu on the main menu bar.
- Refresh – Inspects the SPOT for name and system information.
- System printout – Opens a window to display standard output from the main isolate operating on that SPOT.
- Blink LEDs – Blinks all of the LEDs on the SPOT 10 times, except the power LED. This is useful for finding a particular SPOT among several.
- Reset – Resets the SPOT. It has the same effect as pressing the *Control* button. It restarts the master isolate.
- Synchronize SDK – Downloads the system software from the host workstation's active SDK to the SPOT. This option is only accessible if the SPOT is connected to the host workstation by a USB cable.
- Deploy library – Downloads to the SPOT a fresh copy of the current library in the active SDK. This also removes any downloaded MIDlets and resets the SPOT.

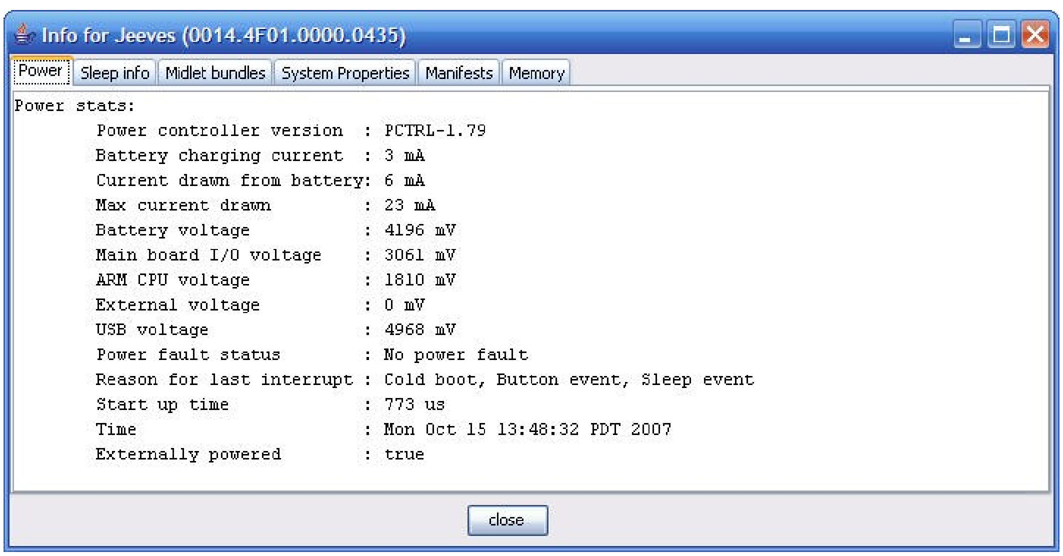
- Deploy application suite – Allows you to specify a project to be deployed to the selected Sun SPOT. The following dialog box displays:



The project directory will optionally be cleaned and compiled. The `jar` file is then deployed to the SPOT, after asking which MIDlet, if any, should be run in the main isolate.

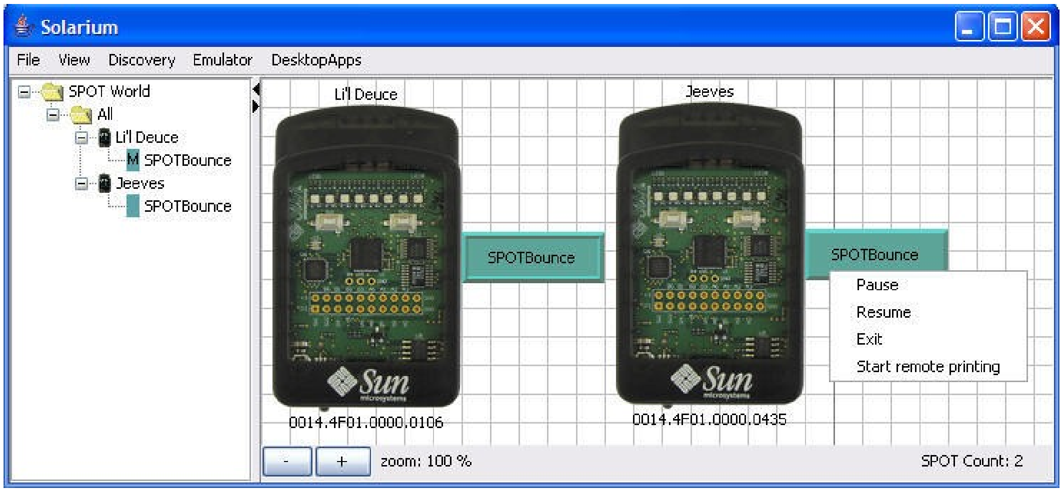


- Get Info – Opens a window containing six tabs. Each tab has information on a different aspect of the selected Sun SPOT.



- Run in Main Isolate – Allows you to specify a MIDlet to run in the main isolate. The system displays a menu of the MIDlets loaded on the SPOT and allows you to pick one. The selected application runs in the main isolate and begins execution the next time the SPOT is reset. Solarium displays the application name to the right of the SPOT in a sunken button display.
- Available Apps – Displays a menu of MIDlets that are available in the deployed jar file. Selecting one causes that MIDlet to start and it adds an application object to the Solarium display for that

SPOT. The application object can be moved around on the graphic display. Clicking the application object gives you a menu for that application.



In the illustration, the SPOTs are both running the Bounce demo. The SPOT on the left is running in the main isolate, as shown by M in the tree view. The SPOT on the right is running it as a child isolate.

The application menu choices are:

- 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 with the Solarium window.

Virtual Sun SPOTs

Solarium also has the ability to display and run applications in virtual Sun SPOTs.

To create a new virtual Sun SPOT in Solarium, pull down the Emulator menu from the main menu bar and select *New virtual SPOT*. Solarium creates a new virtual SPOT and places it in the upper left of the graphic display. It appears 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.

Virtual SPOTs can send and receive radio communications. Their radio communications with physical SPOTs are sent through a shared basestation.

A set of virtual SPOTs can be stored from Solarium. Select “Save virtual configuration” from the “Emulator” pull-down menu in the main menu bar. You are asked to specify a filename and Solarium stores the names of the virtual SPOTs, their IEEE number, their location in Solarium, the `jar` file that was loaded, if any, and the MIDlets that were running on the SPOTs. It also asks you to give a description of the configuration.

The virtual SPOTs in a stored configuration can be reloaded by selecting “Open virtual configuration” from the “Emulator” pull-down menu. Selecting “Display virtual configuration description” displays the description for the configuration most recently loaded.

For a tutorial introduction and more ‘pfinformation about the emulator software, see the *Sun SPOT Emulator Tutorial*, and online HTML document.

Note: In previous releases there was a bug that caused the SPOT emulator to fail if the file path where the Sun SPOT SDK has been installed included a space character. Windows users encountered this problem because the standard installation location for Windows is `C:\Program Files\`. This bug was fixed in the 4.0 Blue release.

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:

- Discover Sun SPOTs – This choice sends a broadcast message, asking Sun SPOTs to identify themselves. If the broadcast message reaches a Sun SPOT running the OTA command server or if the Sun SPOT is connected to the host workstation via USB, the SPOT should be discovered.
- Discover one – This choice allows you to specify an IEEE number for a SPOT. Solarium attempts to find that particular SPOT.
- Set discovery time out – This choice specifies how long Solarium will wait for an answer to its broadcast message. It defaults to three seconds.
- Set discovery hop count – This choice specifies how many radio hops a discovery message may take before it is no longer forwarded from SPOT to SPOT. If using a shared basestation, it must be at least two, since the communication between the main process of the host workstation and the shared basestation counts as one hop. Therefore, at least two hops are required to reach any actual SPOTs. If you wanted Solarium to reach SPOTs that are not within 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:** For a shared basestation, it is not possible to change the radio channel.
- Set radio PAN ID – Allows you to specify the PAN ID used by Solarium during discovery. Defaults to 3. Note: for a shared basestation, it is not possible to change the PAN ID.

Powering a SPOT

The capacity of the built-in battery is 720 milliampere-hours.

The drain of the system varies with use.

Power usage for a typical Sun SPOT unit is shown in the table below.

Processor board state	**Radio**	**Sensor Board**	**Current Draw**
Deep sleep mode[1]	Off	Any	~33 microamperes
Shallow sleep[2]	Off	Not present	~24 milliamperes
Shallow sleep	On	Not present	~40 milliamperes
Awake, actively calculating	Off	Not present	~80 milliamperes
Awake, actively calculating	On	Not present	~98 milliamperes
Shallow sleep	Off	Present	~31 milliamperes
Shallow sleep	On	Present	~46 milliamperes
Awake, actively calculating	Off	Present	~86 milliamperes
Awake, actively calculating	On	Present	~104 milliamperes

1. In deep sleep, the processor and sensor board are both powered down.
2. Shallow sleep means devices active, but no active threads.

Changing the transmit power of the radio effects the current draw slightly. Reducing the transmission power from 0db to -25db results in a savings of about 3 milliamperes.

LEDs also use power. The power draw for a single Sun SPOT LED is specified in the table below.

LED	**Current Draw**
All elements, full brightness	25 milliampere
Blue element, full brightness	10 milliampere
Red element, full brightness	9 milliampere
Green element, full brightness	5 milliampere

The current draw for the LEDs is reasonably linear. An LED at half-brightness draws approximately half the current of an LED at full brightness. Reducing LED brightness to the minimum required for your situation is often a good way to conserve power. Often LED levels of 20, out of a possible 255, are reasonably visible.

The approximate length of time that a full-charged Sun SPOT unit can operate is shown below for most of the conditions of interest.

Sun SPOT State	Battery Life Estimate
Deep sleep	909 days
Shallow sleep, no radio	23 hours
Shallow sleep, radio on	15 hours
CPU busy, no radio	8.5 hours
CPU busy, radio on	7 hours
Shallow sleep, 8 LEDs on, no radio	3 hours

A power fault occurs when one of these conditions occurs:

- External power exceeds 5.5V
- V_{batt} exceeds 4.9V
- V_{CC} ± 10% of 3.0V
- V_{core} ± 10% of 1.8V
- Battery Discharge current exceeds 500ma

A Sun SPOT may also be powered by removing the demo sensor board and supplying power to pins J1 and J2 of the CPU-board top connector. The power should be between 4.5v and 5.5v and at least 1A.

Programming a SPOT

If you are using NetBeans and you have installed the Sun SPOT NetBeans modules, an easy way to create a new SPOT application is to use the *File > New Project* command and select “Sun SPOT Application” from the “General” category. NetBeans then creates all the files you need, including a simple MIDlet that you can modify, and the ancillary files and directory structure that NetBeans and the Ant scripts use.

Alternately, you can just copy a demo project. The copy gets you the ancillary files that NetBeans and the Ant scripts use and it makes sure you get the right subdirectory structure. Finally, there is a sample application in

```
[SunSPOTdirectory]/Demos/CodeSamples/SunSpotApplicationTemplate
```

You can copy that directory to get an extremely simple “Hello, World” application from which you can work. There is also a `SunSpotHostApplicationTemplate` in the same directory that contains a simple application using the basestation.

Note – If you start with one of these templates and rename the main class of the project, you must update the contents of the `[projectdirectory]/resources/META-INF/manifest.mf` file to match.

Sun SPOT applications are just Java applications, so they aren’t hard to program. The main issues for most programmers are (1) debugging the code and (2) determining how to get access to the peripherals on the demo sensor board. We will discuss both in the sections below.

Debugging on a Sun SPOT

You can debug Sun SPOT applications, using either print statements in your code or using the Over The Air (OTA) debugger. The standard output for free range SPOTs can be directed, over the air, to the console of the host workstation.

OTA Debugging

There are three steps to doing OTA debugging. The first step is to enable an OTA link between a Sun SPOT basestation and a free range Sun SPOT. The second is to deploy and run, in debugging mode, the application on the free-range SPOT. The final step is to attach the NetBeans debugger or your debugger of choice to the application and debug the application.

IEEE Extended MAC Address

OTA communication to a Sun SPOT requires the IEEE extended MAC address for all Sun SPOTs involved. The IEEE extended MAC address is a 64-bit address, expressed as four sets of four-digit hexadecimal numbers: `nnnn.nnnn.nnnn.nnnn`. The first eight digits are always `0014.4F01`. The last eight digits should be printed on a sticker visible through the translucent plastic on the radio antenna fin. A typical sticker would say something like `"0000.0106"` and that would imply an IEEE address for that SPOT of `0014.4F01.0000.0106`.

You can also get the IEEE address for a Sun SPOT using the `ant info` command. To get the IEEE address this way, connect your Sun SPOT to the USB cable, open a command window on the host workstation, navigate to any Sun SPOT project directory, and execute the command:

```
ant info
```

You get output that ends with a section that looks like this:

```
[java] Sun SPOT bootloader (red-090621)
[java] SPOT serial number = 0014.4F01.0000.0106
[java]
[java] Startup configuration:
[java]    OTA Command Server is enabled
[java]    Configured to run the this application
[java]       spotsuite://Sun_Microsystems_Inc/BounceDemo-onSPOT/1.0.0
[java]       C:\Program Files\Sun\SunSPOT\Demos\BounceDemo\ExampleJar
[java]       32873 bytes
[java]       last modified Mon Jun 15 12:16:42 PDT 2009
[java]
[java]    Squawk command line:
[java]       -spotsuite://library
[java]       -Xboot:268763136
[java]       -Xmxnvm:0
[java]       -isolateinit:com.sun.spot.peripheral.Spot
[java]       -dma:1024
[java]       -MIDlet-1
[java]
[java] Library suite:
[java]    hash=0xa31883
[java]    Installed library matches current SDK library
[java]    Installed library matches shipped SDK library
[java]    Current SDK library matches shipped SDK library
[java]
[java] Security:
[java]    Owner key on device matches key on host
[java]
[java] Configuration properties:
[java]    spot.battery.model: LP523436B
[java]    spot.external.0.firmware.version: 1.12
[java]    spot.external.0.hardware.rev: 5.0
[java]    spot.external.0.part.id: EDEMOBOARD_REV_0_2_0_0
[java]    spot.hardware.rev: 6
[java]    spot.ota.enable: true
[java]    spot.powercontroller.firmware.version: PCTRL-1.102
[java]
[java] Exiting
-run-spotclient-multiple-times-locally:
-run-spotclient:
BUILD SUCCESSFUL
Total time: 28 seconds
```

The IEEE extended MAC address is the number that follows “SPOT serial number:” In this case, the MAC address is `0014.4F01.0000.0106`.

The IEEE address for a SPOT is also displayed when that SPOT is discovered in Solarium.

Enable an OTA Link

1. **Enable the OTA command server on the free-range SPOT.**

   The OTA command server is enabled on Sun SPOTs direct from the factory. To test whether the OTA command server is enabled, execute the "ant info" as described directly above. The output will say either "OTA Command Server is enabled" or "OTA Command Server is disabled."

   You can enable OTA communication using the SPOTManager tool or a command line. To use the SPOTManager tool, connect the Sun SPOT to the USB cable, go to the Sun SPOTs tab in the SPOTManager, select the Sun SPOT from the pull-down menu, and then click the OTA button and select "Enable."

   To enable the OTA command server using a command line, connect your Sun SPOT to the USB cable, open a command window on the host workstation, navigate to any Sun SPOT project directory, and execute the command:

   ```
   ant enableota
   ```

   If you later decide to disable OTA communication, you can use the command

   ```
   ant disableota
   ```

   You can test whether or not the command worked by repeating the "ant info" command. Disconnect the free-range SPOT from the USB cable.

2. **Enable the SPOT basestation.**

   Connect the basestation SPOT to the USB cable. If you execute an ant info command, in the Startup section, you will see a line that either says:

   ```
   [java] Configured as a Basestation
   ```

   or

   ```
   [java] Configured to run the current application
   ```

   "Configured to run the current application" means that the SPOT is not running in basestation mode.

   To put the Sun SPOT into basestation mode, enter the command:

   ```
   ant startbasestation
   ```

   and the SPOT is put into basestation mode. You can confirm that it is in basestation mode with the `ant info` command. However, the `ant info` command stops the basestation application. After executing the `ant info` command, press the control button on the basestation to restart the basestation application

   Now the free-range SPOT and the basestation are capable of communicating with each other in debugging mode.

Deploy and Run the Application in Debugging Mode

The next step is to deploy the application code to the free-range Sun SPOT and run it in debug mode. To do this:

1. **Open a command window.**

   Under Windows, this is usually available from *Start > All Programs > Accessories > Command Prompt*. On a Macintosh or a Linux machine, any command line window will do.

2. **Navigate to the project directory for the application that you wish to debug.**

3. **Deploy the application to the free-range SPOT using the command:**

   ```
   ant -DremoteId=nnnn.nnnn.nnnn.nnnn deploy
   ```

   where *`nnnn.nnnn.nnnn.nnnn`* is the IEEE extended MAC address for the free-range SPOT. Ant command options are case-sensitive. The options `-DremoteID`, `-DremoteID`, and `-DRemoteId` will not work. It must be `-DremoteId`.

4. **Launch the application in debug mode, using the command:**

   ```
   ant -DremoteId=nnnn.nnnn.nnnn.nnnn debug
   ```

   where *`nnnn.nnnn.nnnn.nnnn`* is the IEEE extended MAC address for the free-range SPOT.

   The command line output rapidly scrolls through some output, then waits for 30 seconds to a minute, and then prints output that ends with:

   ```
   -do-debug-proxy-run:
   [java] Trying to connect to VM on radio://0014.4F01.0000.0106:9
   [java] Established connection to VM (handshake took 70ms)
   [java] Waiting for connection from debugger on serversocket://:2900
   ```

   Note the socket number indicated in the last line.

Attach to the Debugger

The final step is to attach your debugger to the application running on the free-range SPOT. The method for doing this varies with the debugger. For NetBeans, the steps are:

1. **Within NetBeans, open the project that you wish to debug.**

2. **Ask NetBeans to attach the debugger.**

   You can do this either through the *Attach Debugger* command from the Run menu on the main toolbar, or you can press the *Attach Debugger* icon in the upper right. It is a blue triangle, pointing to the right, with a small red square just to the left of it.

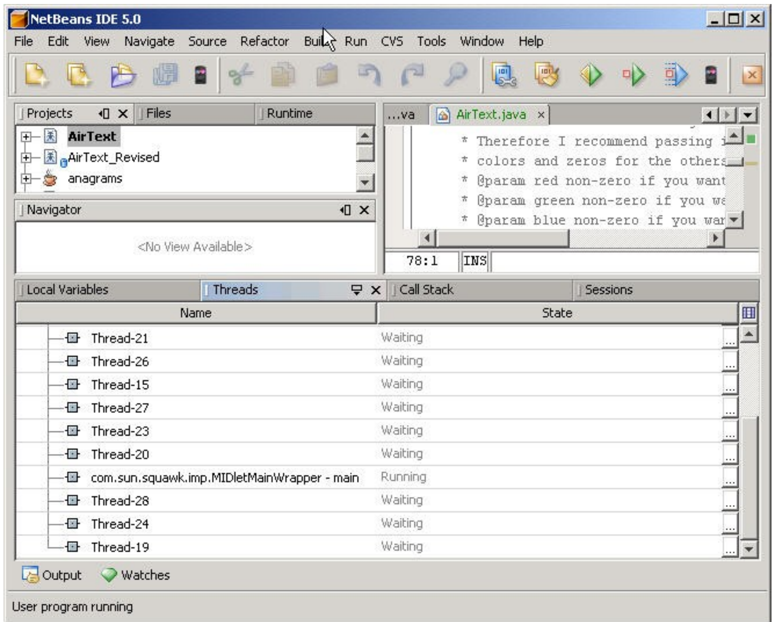
   A dialog box displays.

3. **Specify the debugging attachment.**

   For the connector type, select *SocketAttach*. This gives you an opportunity to enter the port number for the socket. Enter the socket number that you got in command line window after doing the debug-run command. Click *OK*. After about ten or fifteen seconds, the debugger should attach to the application on your free-range SPOT.

4. **Find the application thread and debug.**

   The debugger looks something like this:



   The application thread is the one labeled `MIDletMainWrapper`.

It is beyond the scope of this document to explain NetBeans and debugging commands, but NetBeans has a good help system and has a typical set of debugging commands.

Print Debugging

For print debugging, load the application to be debugged onto a Sun SPOT. As you debug, add calls to `System.out.println()`. For example:

```
System.out.println("Got to the Foobar method call");
```

Start the application using either the `Run` command from within NetBeans or by issuing an `ant run` command at command-line prompt from within the project directory. If the SPOT is not connected to the host workstation, the IEEE address of the SPOT must be included in the `ant` command:

```
ant -DremoteId=nnnn.nnnn.nnnn.nnnn
```

where *`nnnn.nnnn.nnnn.nnnn`* is the IEEE address of the SPOT.

Unless they have been redirected `System.out` and `System.err` appear on the host workstation. In NetBeans, it appears in an Output panel, normally along the bottom of the NetBeans window. In a command-line window, the debugging output appears as part of the output from the `ant run` command.

The process that runs Sun SPOT applications from the host produces a fair amount of output to the command line or output window. When debugging, be sure to scroll back through that output to see if anything important has been printed but scrolled out of sight.

Accessing the Sensor Board

The sensor board includes a 3D accelerometer, a temperature sensor, a light sensor, eight LEDs, two switches, five general-purpose I/O pins, and four high current output pins. In this section, we give a rapid introduction to using these components in a Sun SPOT Java program. For more details, see the Javadoc pages in the `[SpotSDKdirectory]/doc/javadoc` directory and see the demonstration applications in the `[SpotSDKdirectory]/Demos/CodeSamples` directory. These applications are simple working applications that show the details of using one or two sensor board devices.

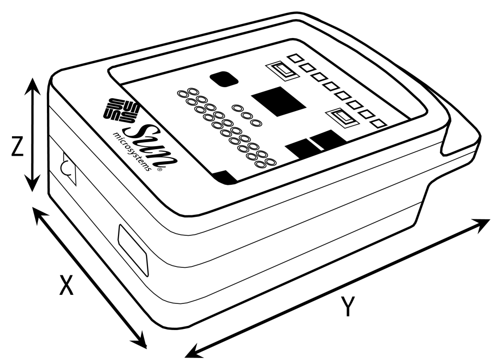
The simplest interfaces to sensor board devices are also described in the sections below. All of the sensor board input devices also have listener classes associated with them. These listener classes are detailed in the Javadoc.

Accelerometer

This is a very brief introduction to the Sun SPOT accelerometer. For more detail, see the AppNote included in `[SpotSDKdirectory]/doc/AppNotes/AccelerometerAppNote.pdf`.

There are three axes on which the accelerometer measures acceleration. The Z-axis is perpendicular to the Sun SPOT boards. The X-axis is parallel to the row of LEDs on the sensor board. The Y-axis is parallel to the long edge of the sensor board.

The accelerometer's X, Y, and Z axes are illustrated below.



In the figure above, the plus (+) on the end of an axis indicates that when the device's acceleration vector increases in that direction, the associated accelerometer readings grow larger.

To use the accelerometer:

```
//Create an accelerometer interface instance
import com.sun.spot.sensorboard.EDemoBoard;
import com.sun.spot.sensorboard.IAccelerometer3d;
IAccelerometer3D ourAccel = EDemoBoard.getInstance().getAccelerometer();
//Read from the accelerometer
double x-accel = ourAccel.getAccelX();
double y-accel = ourAccel.getAccelY();
double z-accel = ourAccel.getAccelZ();
```

The readings are in g-force units. There is also a method `getAccel()` that returns the vector sum of the acceleration along all three individual axes.

The accelerometer interface also has methods for determining the acceleration relative to a previously set acceleration. This allows you to remove the force of gravity from your measurements.

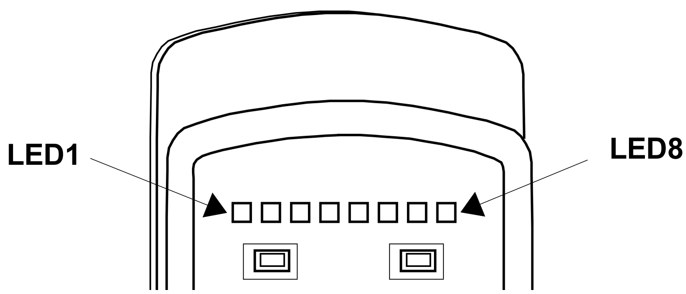
```
// Zero out the current forces, usually gravity
ourAccel.setRestOffsets();
// see if we are accelerating up or down
double z-relative-accel = ourAccel.getRelativeAccelZ();
```

`IAccelerometer3D` also has methods that calculate the orientation of the SPOT to the acceleration of the SPOT. When the SPOT is at rest, this acceleration is gravity and the tilt is relative to gravity. The methods are `getTiltX()`, `getTiltY()`, `getTiltZ()`, and they return the tilt in radians.

More details are available in the Javadoc on `IAccelerometer3D` class. If you require more control than is available through this interface, look at the Javadoc for the `LIS3L02AQAccelerometer` class.

LEDs

There are eight three-color LEDs on the demo sensor board, in a row, with LED1 on the left and LED8 on the right.



Each LED has a red, a green, and a blue emitter as part of the LED. Each individual color can have an intensity from 0 to 255, with 0 being off and 255 being as bright as possible.

To use the LEDs:

1. **Instantiate the LED object array.**

```
Import com.sun.spot.sensorboard.EDemoBoard;
Import com.sun.spot.sensorboard.ITriColorLED;
ITriColorLED[] ourLEDs = EDemoBoard.getInstance().getLEDs();
```

2. **Set the LED color desired.**

Colors are specified with the `setRGB(int red, int green, int blue)` method.

```
// set the LED color desired
// set the first two LEDs to bright red, the next two to bright green,
// the next two to bright blue, and the last two to white.

// First two = bright red
ourLEDs[0].setRGB(255,0,0); ourLEDs[1].setRGB(255,0,0);

// Next two = bright green
ourLEDs[2].setRGB(0,255,0); ourLEDs[3].setRGB(0,255,0);

// Next two = bright blue
ourLEDs[4].setRBG(0,0,255); ourLEDs[5].setRGB(0,0,255);

// Last two = white
ourLEDs[6].setRGB(255,255,255); ourLEDs[7].setRGB(255,255,255);
```

3. **Turn the LEDs on.**

```
//turn the LEDs on
for (int i = 0; i < 8; i++){
   ourLEDs[i].setOn()
}
```

4. **If desired, turn the LEDs off.**

```
// turn the LEDs off
for (int i = 0; i < 8; i++){
   ourLEDs[i].setOff()
}
```

You can also query the state of the LEDs using the `isOn()`, `getRed()`, `getGreen()`, and `getBlue()` methods.

Switches

The sensor board has two switches on it. These are represented in the `EDemoBoard` object as an array of type `ISwitch`. You may query the state of the switches using the `isOpen()` and `isClosed()` methods. Ordinarily you implement an event loop that checks the switches used in your application on a periodic basis, or you ask the Sun SPOT to stop and wait for the switch state to change. When you want the SPOT to wait for the state switch to change, you would use the `waitForChange()` method.

1. **Instantiate the switch array.**

```
Import com.sun.spot.sensorboard.EDemoBoard;
Import com.sun.spot.sensorboard.ISwitch;
ISwitch[] ourSwitches = EDemoBoard.getInstance().getSwitches();
```

2. **Look for a switch press.**

If you wanted a switch press and were willing to wait for it:

```
if(ourSwitches[0].isOpen()){
// if it is open, wait for it to close
   ourSwitches[0].waitForChange();
}
// Whether it was closed before or just closed, wait for it to open
ourSwitches[0].waitForChange();
```

Light Sensor

The light sensor returns an integer that ranges from 0 to 750. Zero represents complete darkness. Peak sensitivity of light sensor is at 600nm wavelength. An illustration of how the raw readings map to luminance (lx) values is given in the table below:

Luminance	Raw Reading
1000 lx	497
100 lx	50
10 lx	5

To use the light sensor:

1. **Instantiate a light sensor object.**

```
Import com.sun.spot.sensorboard.EDemoBoard;
Import com.sun.spot.sensorboard.peripheral.ILightSensor;
ILightSensor ourLightSensor = EDemoBoard.getInstance().getLightSensor();
```

2. **Get the light sensor raw reading.**

```
int lightSensorReading = ourLightSensor.getValue();
```

This is fine for a constant light source. However, some light sources, specifically fluorescent light bulbs, while seeming constant to the human eye, actually vary rapidly. For these sources, it is better to use the method `getAverageValue(int n)`. The method returns the average of `n` samples taken at 1 millisecond intervals. If n is not specified, 17 samples are taken, spanning one sixtieth of a second, or the usual length of a power/light cycle.

```
int lightSensorReading = ourLightSensor.getAverageValue(34);
```

Temperature Sensor

The temperature sensor is the simplest of the sensors. There are no raw readings and no parameters to set. However, it is, inevitably, close to some heat sources in the Sun SPOT. More accurate temperature readings could be obtained with an external temperature sensor tied to the I/O pins on the sensor board.

1. **Instantiate the temperature sensor object.**

```
Import com.sun.spot.sensorboard.EDemoBoard;
Import com.sun.spot.sensorboard.io.ITemperatureInput;
ITemperatureInput ourTempSensor = EDemoBoard.getADCTemperature();
```

2. **Read the temperature.**

```
// The temperature can be read in Celsius
double celsiusTemp = ourTempSensor.getCelsius();
// or in Fahrenheit
double fahrenheitTemp= ourTempSensor.getFahrenheit();
```

Radio Communication

The `RadiostreamConnection` interface provides a socket-like peer-to-peer radio protocol with reliable, buffered stream-based IO between two devices. This communication can be single-hop or multi-hop.

To open a connection:

```
StreamConnection conn = (StreamConnection)
Connector.open("radiostream://nnnn.nnnn.nnnn.nnnn:xxx");
```

where *`nnnn.nnnn.nnnn.nnnn`* is the 64-bit IEEE address of the radio, and *`xxx`* is a port number in the range 0 to 255 that identifies this particular connection. Note that 0 is not a valid IEEE address in this implementation.

To establish a connection both ends must open connections specifying the same port number and complimentary IEEE addresses.

Once the connection has been opened, each end can obtain streams to use to send and receive data. For example:

```
DataInputStream dis = conn.openDataInputStream();
DataOutputStream dos = conn.openDataOutputStream();
```

Radiograms

The `RadiogramConnection` interface defines the "radiogram" protocol – the radiogram protocol is a datagram-based protocol that allows the exchange of packets between two devices.

To establish a point-to-point connection both ends must open connections specifying the same `portNo` and corresponding IEEE addresses. Port numbers between 0 and 31 are reserved for system services. Use of these ports by applications may result in conflicts.

Once the connection has been opened, each end can send and receive data using a datagram created on that connection. For example:

```
...
DatagramConnection conn = (DatagramConnection) Connector.open("radiogram://" +
targetIEEEAddress + ":100");
Datagram dg = conn.newDatagram(conn.getMaximumLength());
dg.writeUTF("My message");
conn.send(dg);
...
conn.receive(dg);
String answer = dg.readUTF();
...
```

The radiogram protocol also supports broadcast mode, where radiograms are delivered to all listeners on the given port. Because broadcast mode does not use 802.15.4 ACKs, there are no delivery guarantees.

```
...
DatagramConnection sendConn = (DatagramConnection)
Connector.open("radiogram://broadcast:100");
```

```
dg.writeUTF("My message");
sendConn.send(dg);
...
```

The radiogram protocol also supports server mode, where any radiogram sent on the given port will be received. These may be broadcast packets of packets specifically addressed to the receiving SPOT.

```
...
DatagramConnection recvConn = (DatagramConnection)
Connector.open("radiogram://:100");
recvConn.receive(dg);
String answer = dg.readUTF();
```

Unicast datagrams have the hop count determined as part of route discovery. Broadcast datagrams are sent for the number of hops specified by the connection. The default value is 2 hops. To make a single hop:

```
sendConn = (RadiogramConnection) Connector.open (
     "radiogram://broadcast:100");
sendConn.setMaxBroadcastHops(1);
```

Broadcast mode is not recommended for datagrams larger than 200 bytes. Because the list of recipients is unknown, broadcast mode is inherently unreliable. With SPOTs, there is also a known problem that may result in fragments of a datagram becoming lost.

The maximum broadcast packet size is 1260 bytes of payload. An individual 802.15.4 radio packet only carries about 100 bytes of data, though the amount varies slightly depending on whether the packet has a mesh header and whether or not the fragment is the first fragment in the datagram.

Broadcast datagrams that result in two fragments are fairly reliable. Datagrams broken into three fragments (over 200 bytes of payload) are likely to experience some loss. Broadcasts datagrams broken into more than three fragments will almost certainly see some loss. The problem is related to the ability of the receiving side to empty the incoming buffer fast enough. This problem may be exacerbated by a receiver side that frequently garbage collects or has a large number of active threads.

Important data should generally be unicast via radiograms or radiostreams. The inherent ACK/retry mechanism of radiograms generally insures either delivery or notification of failure. Likewise, radiostreams provide automatic fragmentation and an additional level of assurance that fragments are reassembled in the proper order.

Alternatively, if broadcast is required, the application should attempt to limit packet size so that each broadcast results in less than 3 fragments. Single packet broadcasts result in more data space as a fragmentation header is not required. Additionally, inserting a 20ms pause between the sending of broadcast packet assists in allowing the receiver to keep up with packet reception.

Troubleshooting

Software, All Platforms

Problem – But I don't want to use NetBeans!

One of the things that NetBeans does is set some needed environmental variables. If you don't use NetBeans, do the following:

- Modify your `PATH` to include the bin subdirectory for your Java Development Kit (JDK).
- Modify your `PATH` to include bin subdirectory for the Ant directory.
- Set `JVMDLL` to the location of your Java virtual machine (Windows only).
- Set `JAVA_HOME` to the location of your top level Java SDK directory (Windows and Linux).

NetBeans also provides a user interface for the several `Ant` commands used to deploy and run code on the Sun SPOTs from the host workstation. Read through the Sun SPOT Developer's Guide and make sure you understand how to use the Ant commands described there.

Problem – You get the following error message any time you attempt to communicate with a Sun SPOT over the USB cable, including any of the many ant commands:

```
[java] WARNING:  RXTX Version mismatch
[java] Jar version = RXTX-2.1-7
[java] native lib Version = RXTX-2.1-7pre17
```

The host workstation uses a serial communication package called RXTX to communicate with the Sun SPOTs. The version in use on the host workstation and on the Sun SPOT must match. The correct version of the RXTX library is installed on the host workstation when the SDK is installed, but if another version is on the load path, a mismatch can occur. Look for an old version of the RXTX library somewhere in your load path. The name of the RXTX file varies with the operating system of the host workstation, as shown in the table below.

Operating System	RXTX File Name
Windows	`rxtxSerial.dll`
Macintosh	`librxtxSerial.jnilib`
Linux	`librxtxSerial.so`

Change your `PATH` variable to avoid the other version of RXTX or remove the excess version of RXTX.

Problem – My Sun SPOT has got into a state where it continually restarts and I get an error whenever I try to deploy to it. How can I recover?

First you need to get the SPOT into a state where it is listening to commands from the host rather than continually restarting. To do that, follow this procedure:

- Disconnect the Sun SPOT from the USB cable.
- Kill all the `ant` and `java` processes listening on the port.
- Hold the control button in for a few seconds until a double red flash indicates that the Sun SPOT has powered down.
- Type this command at a command prompt:

```
ant -Dport=COMnn info
```

substituting the correct communication port/device name for `COMnn`.

If you don't know the correct port name just enter a dummy value, e.g., `"X"`, and complete the procedure. The output should list the correct port name. Then repeat the procedure with the correct port.

As soon as the `ant` script starts to complain that the port isn't available, saying something like:

```
[java] Port COM31 unavailable...
[java] Available ports: COM1 COM2 COM3 LPT1
[java] retrying...
```

immediately plug in the Sun SPOT.

The `ant info` command should now operate correctly. When it has finished, you can take the necessary recovery action, normally to reinstall the system software using `ant upgrade` and then redeploying your (possibly corrected) application.

Problem – When I try to create a suite (via `ant suite` or `ant deploy`) I get the following error:
`Unable to locate tools.jar Expected to find it in C:\...`

Most likely your system environment variables are not setup correctly. Please make sure your PATH variable has your JDK directory as its FIRST value. This also ensures that Windows is not using the `java.exe` commonly located in `C:\Windows\System32`. Also ensure that `JAVA_HOME` is set correctly.

Problem – How do I modify the `sunspot.home` property for the `ant` build system?

You need to modify the file `.sunspot.properties` located in your home directory. In Windows, this is `C:\documents and settings\(your account)`.
On the Macintosh, this is `/Users/(your account)`.

Problem – Is there an API to query the ID of a SPOT at run-time?

```
System.getProperty("IEEE_ADDRESS").
```

Problem – When I do an OTA ant deploy, I get a "No route found" error.

If your attempt to do an OTA ant deploy shows console output that looks like:

```
-run-spotclient-once:
[java] SPOT Client starting...
[java] Error: No route found
[java] The SPOT client will now exit
BUILD FAILED.
```

then you probably have the OTA command server enabled on your basestation. It should not be so enabled. Do an `ant disableota` with the basestation as the target and retry your OTA deploy.

Software, Linux

Problem – *(Linux):* When trying to deploy an application to a Sun SPOT, I get error messages from RXTX, saying that the device is not available, or that it doesn't have permission to access it or the lock file.

Make sure that you followed the instructions in the section "Adjust Permissions (Linux)" on page 29 of the *Installation Instructions*. In particular don't forget to log out after performing the changes.

If you still cannot deploy to the Sun SPOT, make sure that the `/var/lock` directory exists, and that it is has read, write, and execute permissions for the group to which you added the user. If it doesn't exist, you should create it. The recommended permissions are `755`, and the group should be lock. Also check the permissions and group of the device file. The name of the device should be in the error message; common names are `/dev/ttyACMx` or `/dev/usbmodem`.

If this doesn't fix the problem, verify that there is no link to `/var/lock` or `/var/spool/lock` as this may cause RXTX to detect the lock file twice and to fail.

Problem – *(Linux):* When trying to deploy an application to a Sun SPOT I get an error message: "No Sun SPOT devices found. ...." or similar.

In addition to the obvious things, like making sure that a Sun SPOT is actually connected to the USB port, it may be that your Linux installations doesn't have the `cdc_acm` driver that is required to access the Sun SPOT. Try calling `"dmesg | grep usb"` in a command window. There should be a message that looks something like `"usb ...: New full speed USB using ..."`. Furthermore there should also be a message referring to `cdc_acm`. If you see the general messages but you do not see any `cdc_acm` messages, you probably have to install the `cdc_acm` driver onto your system.

If you see the `cdc_acm` messages, and still get a "No Sun SPOT devices found..." error, it may be that the spot-finder script has failed to detect the device. This may occur if the `hal_device` command is not available and the device gets assigned a name not matching the expected `/dev/ttyACM*` pattern. We recommend you install `hal_device` on your system in this case. Alternatively, you may specify the appropriate device name manually in the `ant port` property. You define this property in the `build.properties` for a project or in the command line to `ant`.

Problem – (Linux): When trying to deploy an application to a Sun SPOT, I get an error message:
`Port Error: An SELinux policy prevents this sender from sending this message to this recipient unavailable...`
or something similar.

This means that the SELinux policy is too restrictive and is not allowing access to the serial device. This is, for example, the case in the default settings for Fedora Core 5.

If you don't require the additional security, SELinux allows you to either disable it or set it to a less restrictive setting. For example, in Fedora Core 5, changing the SELinux setting to permissive solves this problem. You can change the SELinux setting in the `System/Administration/Security Level and Firewall` menu. If you don't want to give up this security level, you need to define a SELinux policy that gives RXTX permission to the device.

Spots in General, Hardware

Problem – SPOT doesn't power up. Neither the power LED nor the activity LED are lit.

Check the battery connector.

Plug the SPOT unit into a powered USB host. This can be a powered USB hub, a USB port on Computer, or a USB mini Type B charger. Make sure the connectors are seated properly and the host device is powered. If the SPOT is not a basestation SPOT and if power LED starts to slowly flash green, then the battery is charging; allow the battery to charge for three hours.

Push the attention button, do not hold it in.

Additional Troubleshooting

If you have a digital voltmeter and are comfortable using it on small circuits, make the following voltage measurements. Unscrew the main retaining screw (Phillips head) and remove the eDEMO board to expose the main board. Leave the battery installed and plug the USB port into a powered source. Set the DVM to measure volts and connect the leads of the DVM to common and volts. Connect the common (-) lead to gold ring around the screw retention hole. From the top of the board with the antenna at the top, find a via marked VSTBY. This is above and to the right of the 30 pin white connector. This should measure +3.0V ±5% when any power source is connected. Find the left most ceramic capacitor next to 3V. This is on the lower left side of the board. Put the positive (+) lead on the bottom of this capacitor. This should measure +3.0V ±5% when power is on. Find the capacitor below the 3.0V marked 1.8V. On the top lead of the capacitor measure 1.8V ±5%. If the SPOT is not a basestation perform the following measurement. In the lower right hand corner is a through hole pad marked V+, this should measure between 3.2V and 4.7V. If you remove the USB, it should not drop below 3.2V. If it does, the battery is defective or dead.

Problem – The Sun SPOT powers up but doesn't boot. The power LED comes on green but the activity LED doesn't flash.

Connect the Sun SPOT to the USB port of your host workstation and run `ant info` to see if the problem is a faulty activity LED.

The Sun SPOT has either a hardware fault or a corrupt ARM bootloader. This requires factory service.

Problem – The Sun SPOT powers up and the power LED turns constant bright red. This may occur during upgrading of power controller firmware.

This is corrupted firmware for the power controller. Try to upgrade without power cycling the SPOT. This may require factory service to restore.

Problem – The Sun SPOT comes on and boots (activity LED flashes green) but host workstation can't find the Sun SPOT as a USB device.

Check the USB cable. Check for potential interfering drivers on the host workstation (try on a different computer). Exit any application that is trying to communicate to the Sun SPOT, disconnect the USB cable, hold the attention button for > 3 seconds, then tap the attention button to restart the SPOT. Plug the cable back in and try again.

For Windows only: Look in the Device Manager to see if there is an "unknown" USB device. If there is, see if it appears and disappears when you connect and disconnect the Sun SPOT. If the unknown device appears to be the SPOT, connect it, ask Windows to uninstall it, then disconnect the SPOT, then reconnect it. You should now get the "new device" dialog and be able to reinstall the driver.

Problem – The Sun SPOT comes on and starts to boot, but the activity light flashes red and not green.

The ARM9 memory test has failed. Retry, but if there are continued failures this indicates bad memory and requires factory service.

Problem – The Sun SPOT comes on and boots. The host workstation USB finds the SPOT but says the device is busy.

There is probably an active process still connected to the SPOT. Disconnect the Sun SPOT, kill all the processes that talk to the SPOT, reboot the SPOT and reconnect.

Problem – The Sun SPOT continually reboots.

The Sun SPOT may have encountered an exception that causes it to continually reboot. There may be a problem in the deployed SPOT application. Do `ant echo` and look at any output from the SPOT to see what the problem is. Try deploying a known demo to the SPOT and see if that works. If not try reloading the SPOT system code by running `ant upgrade` on the SPOT.

Problem – The Sun SPOT communicates with the host workstation USB but the power LED flashes red.

Connect the Sun SPOT to the host workstation USB. Load the application in the demo folder called `SpotCheck`. This prints out the voltage and current usage of the SPOT along with any fault issues with the SPOT.

Problem – The Sun SPOT communicates with the host workstation USB but the eDEMO board is not working properly.

Load and run the application in the `demo` folder called `SpotCheck`. Cycle through the tests to check the light sensor, temperature sensor, accelerometer, and LEDs on the eDemo board.

Problem – The Sun SPOT communicates with the host workstation USB but doesn't communicate with the radio.

Check for potential interference and shielding of the antenna fin. Antenna (thin part of the SPOT box) must be clear of metal objects. Testing requires two SPOTs with eDEMO boards. Load and run an application in the demo folder called RadioStrength. Each SPOT then takes turns transmitting and receiving data packets and the signal strength is displayed as a bar graph on the Sun SPOT LEDs.

If Your Sun SPOT Needs Factory Service

If your Sun SPOT requires factory service, please send an email to `info@sunspotworld.com` with "Service request" in the subject line of the message. Please summarize the problem with your Sun SPOT in the body of the message.

Battery Warnings

Do not short-circuit battery. A short-circuit may cause fire, explosion, and/or severe damage to the battery.

Do not drop, hit or otherwise abuse the battery as this may result in the exposure of the cell contents, which are corrosive.

Do not expose the battery to moisture or rain. Keep battery away from fire or other sources of extreme heat. Do not incinerate.

Exposure of battery to extreme heat may result in an explosion.

No other battery substitutions or different chemistry batteries should be used.

Do not bypass the battery protection circuit.

Dispose of batteries properly. Do NOT throw these batteries in the trash. Recycle your batteries, if possible.

Federal Communications Commission Compliance

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try and correct the interference by one or more of the following measures: Reorient or locate the receiving antenna. Increase the separation between the equipment and receiver. Connect the equipment into an outlet on a circuit different from that to which the receiver is connected. Consult the dealer or an experienced radio/TV technician for help.

The Sun SPOTs are supplied with a shielded USB cable. Operation with a non-shielded cable could cause the Sun SPOTs to not be in compliance with the FCC approval for this equipment. The antenna used with this transmitter must not be co-located or operated in conjunction with any other antenna or transmitter; to do so could cause the Sun SPOTs to not be in compliance with the FCC approval for this equipment. Any modifications to the Sun SPOTs themselves, unless expressly approved, could void your authority to operate this equipment.

FCC Declaration of Compliance:

Responsible Party:
Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
Phone: US 1-800-555-9786; International 1-650-960-1300

Products:
SLS-E5-XXXX
where “X” is any alphanumeric character or a blank.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following conditions: this device may not cause harmful interference and this device must accept any interference received, including interference that may cause undesired operation.

This device can be used as is (stand-alone) or as a module (part of a final host product). If the device will be used as a module these rules must be followed:

1. **Integrator must place a label outside their product similar to the example shown:**

OEM manufacturer name Contains transmitter module FCC ID: UDM3011 UPN: 1894B-3011 Model: 3011

2. **Caution: Exposure to Radio Frequency Radiation**

    To comply with FCC RF exposure compliance requirements, a separation distance of at least 20 cm must be maintained between the antenna of this device and all persons. This device must not be co-located or operating in conjunction with any other antenna or transmitter.

    Module 3011 and antenna tested with must be integrated in the end product in such a way that the end user cannot access the either the module, cables, or antennas.

    The installer of this radio equipment must ensure that the antenna is located or pointed such that it does not emit RF field in excess of Health Canada limits for the general population; consult Safety Code 6, obtainable from Health Canada's website `www.hc-sc.gc.ca/rpb`.