Imote2 for Structural Health Monitoring: User's Guide

Illinois Structural Health Monitoring Project

January 2011

ILLINOIS STRUCTURAL HEALTH MONITORING PROJECT
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
OPEN SYSTEMS LABORATORY & SMART STRUCTURES TECHNOLOGY LABORATORY

Table of Contents

1 Introduction

This *User's Guide* is designed for the general users who want to build basic structural health monitoring (SHM) application with wide functionality of ISHMP Services Toolsuite. This guide will allow you to:

- Have a list of required and optional hardware
- Compile and program the motes
- Acquire acceleration data from a network of Imote2s
- Utilize sleep mode for power saving
- Achieve an autonomous network operation
- Deploy the Imote2 on full-scale structures
- Trouble-shoot the network

Comprehensive guides for installation of TinyOS and the ISHMP Services Toolsuite are available on the Illinois Structural Health Monitoring Project (ISHMP) website at: http://shm.cs.uiuc.edu/documentation.html. The first part of the Getting Started for Advanced Users and Developers guide walks you through the process of setting up the PC environment required to interface with the Imote2. The second part of the Advanced Users guide walks you through the installation of the ISHMP services Toolsuite. This *User's Guide* assumes that both TinyOS and the ISHMP Services Toolsuite have been successfully completed.

There are four main parts to this guide. Chapter 2 describes the equipment required for setting up a network of Imote2s, including battery board and sensor board hardware. Chapter 3 gives instructions for the use and functionality of the basic SHM software available in the ISHMP Services Toolsuite. Chapter 4 provides information on the necessary items and steps required to achieve a deployment of Imote2s. Chapter 5 is a general troubleshooting guide for commonly encountered problems (not specific to any particular piece of software). At the end of the guide there is a glossary of terms for added clarification.

Comments and questions:

If you have questions about the software or this guide, or run into problems, please join us on the Imote2 discussion forum: http://vibration.shef.ac.uk/imote2_forum.

2 Equipment/Parts

Each smart sensor node consists of an Imote2 that provides the radio and processor, a sensor board that provides the sensing capability, and a battery board that provides an interface between the power source (batteries) and the Imote2. The Figure 1 shows the top and bottom view of the Imote2 (left), the Imote2 stacked on a battery board with an external antenna (middle), and a sensor board with the Imote2 (right). The use of an external antenna is optional; see Chapter 4 for information on antenna selection and configuration. Table 1 provides a list of the items needed to create a network of Imote2s.

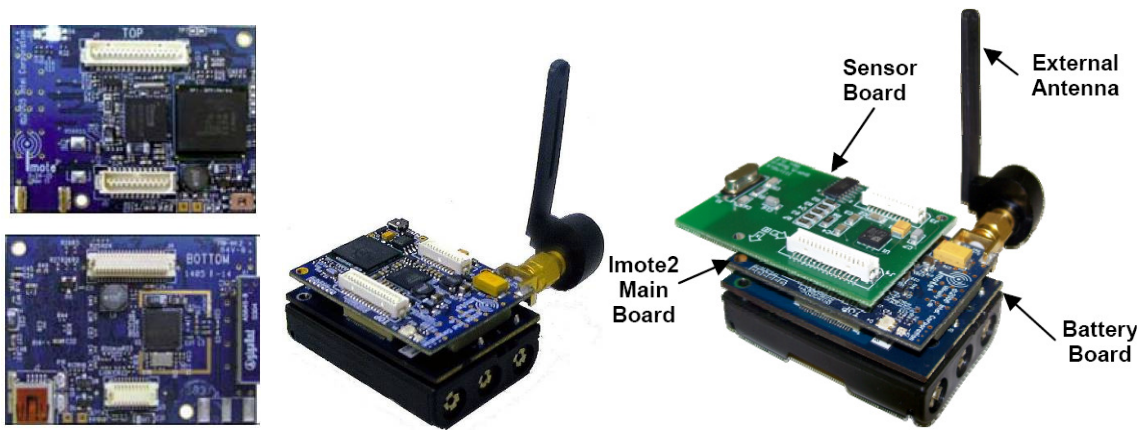


Figure 1. Composition of smart sensor node.

Table 1. Required components for programming and using a network of Imote2s

Item	Source/Vendor	Part No.
Imote2	MEMSIC	IPR2400[1]
Battery board	MEMSIC	IBB2400[2]
USB A to USB mini-B cable	MEMSIC or others	
Debug/Interface Board	MEMSIC	IIB2400
ISM sensor board OR ITS sensor board	MEMSIC	ISM400 ITS400

[1]There are two versions of the Imote2 available from MEMSIC. DO NOT order the .NET edition (IPR2410), which comes preloaded with software that is incompatible with the ISHMP software.
[2]Battery boards come with the Imote2s when purchased from MEMSIC.

You will need at least two Imote2s to create a "network." One Imote2 connects to the PC via USB and acts as the gateway between the PC and your network of remote sensors.

In this guide, the Imote2 that is connected to the PC will be referred to as the *gateway node* and the nodes that make up the network will be referred to as the *leaf nodes* (see Figure 2).

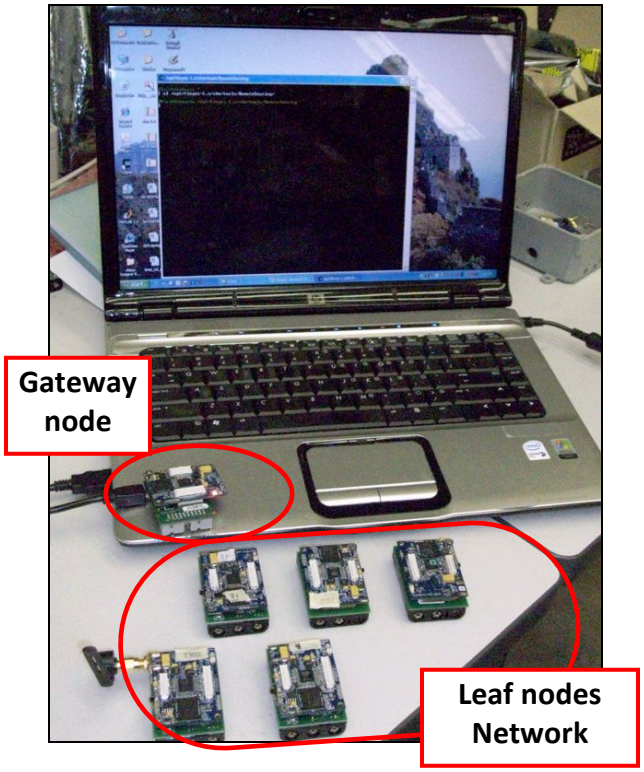


Figure 2. Sample network.

The interface/debug board (IIB, pictured in Figure 3 without (left) and with (right) the Imote2) is required for data collection from the gateway node to the PC. It provides two serial port (UART) interfaces over the USB connection to your PC, one of which communicates debug commands and output, and the other which communicates data.

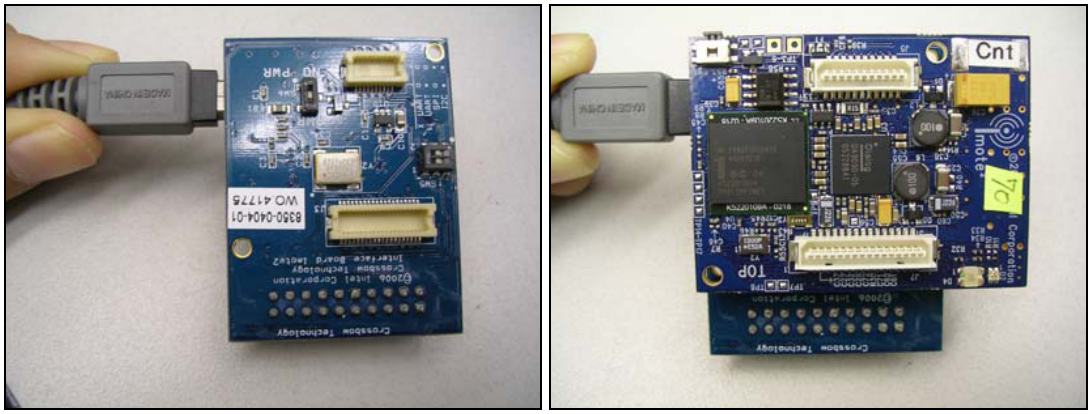


Figure 3. Interface board (IIB2400).

The Imote2s are programmed via USB. Please refer to the Getting Started for Advanced Users and Developers guide for more information regarding how USB and the IIB are used to program/interact with the Imote2 (http://shm.cs.uiuc.edu/documentation.html).

2.1 Sensor Board Options

There are two options for measuring acceleration with the Imote2. One is the ITS400 sensor board (pictured in Figure 4, left). The second is the ISM400 (formerly SHM-A) sensor board (pictured in Figure 4, right), which was developed at the University of Illinois at Urbana-Champaign. Both sensor boards are available from MEMSIC and supported by the software provided in the ISHMP Services Toolsuite. In addition to performance differences, the sensor boards support different sampling rates and have different minimum battery requirements.

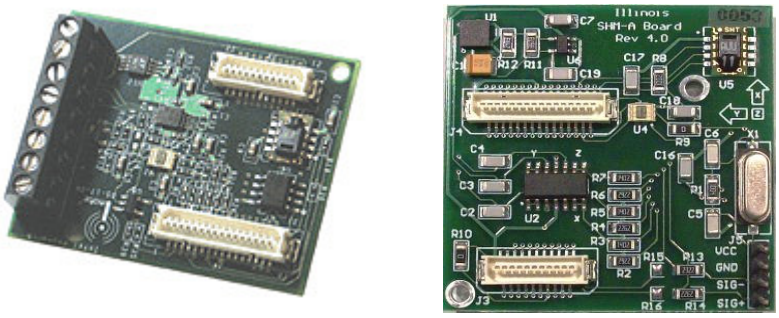


Figure 4. Sensor board: ITS400(left), ISM400(right).

The nominal sampling rates for the ITS400 sensor board are shown in Table 2. The minimum battery voltage (when using the IBB2400 battery board) is 3.6V. More information on the ITS400 can be found on the MEMSIC website at http://memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=137%3Aits400.

Table 2. Supported sampling rates for the ITS400 sensor board[1].

Sample Rate (Hz)	Digital cut-off frequency (Hz)
280 (40)	70 (10)
560 (160)	140 (40)
1120 (640)	280 (160)
4480 (2560)	1120 (640)

[1]From the data sheet for the accelerometer on the ITS400 sensor board: http://www.st.com/stonline/products/literature/ds/10175.pdf.

The default sampling rates supported by the ISHMP software for the ISM400 sensor board are given in Table 3. Additional sampling rates and channel configurations may be created. Please refer to the ISM400 Advanced User's Guide for instructions on creating new parameters. The minimum battery voltage required for the ISM400 sensor board is 3.7V (when using the IBB2400 battery board). More information can be found in the ISM400 Datasheet and User's Guide (http://shm.cs.uiuc.edu/hardware.html).

Table 3. Supported default sampling rates for the ISM400 sensor board[1].

Sample Rate (Hz)	Digital cut-off frequency (Hz)
25	10
50	20
100	40
280	70

2.2 Battery Board

The battery board that come with the Imote2s from MEMSIC take three AAA batteries connected in series. The nominal voltage of one AAA battery is 1.5V, thus the nominal voltage supplied by three batteries is 4.5V. Typical new AAA batteries actually start out with higher than the nominal voltage output. The result is that three new AAA batteries can exceed the maximum battery voltage allowed by the IBB2400 battery boards (below, left) of 4.7V. If this happens, a safeguard mechanism on the battery board will stop it from supplying power to the Imote2. In this case, it may be necessary to use one slightly used battery with two newer batteries ensure the voltage is below the maximum allowed. The voltage coming from the batteries may easily be tested with a voltmeter as shown in Figure 5.

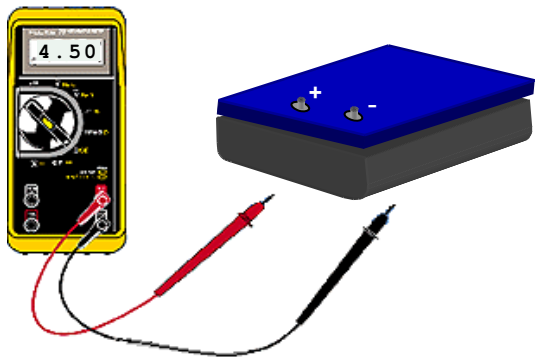


Figure 5. Battery voltage check with a voltmeter.

3 Software

This guide has been written for use with the open-source software available through the Illinois Structural Health Monitoring Project (ISHMP) at http://shm.cs.uiuc.edu/. This Toolsuite provides a library of customizable services for creating a range of SHM applications for networks of smart sensors. In addition, the Toolsuite provides a number of tools and utilities to aid in the development of deployable applications. This guide will focus on the tools, utilities, and applications within the Toolsuite that enable the acquisition of high-fidelity data from a network of Imote2s while ensuring robust network functionality. Specifically, you will do the following:

- Acquire synchronized acceleration data with varying sensing parameters from a network of sensors using the *RemoteSensing* application
- Store sensed data on the PC using the *imote2comm* tool
- Enable a sleep/wake cycle on the network using the *SnoozeAlarm* tool to improve power consumption
- Create an autonomous network that detects when a threshold has been exceeded and wakes the network to collect data using the *AutoMonitor* and *ThresholdSentry* applications
- Automatically create data and log files using the *autocomm* tool

There are three main sections to the software instructions: 1) *RemoteSensing* without *SnoozeAlarm*, 2) *RemoteSensing* with *SnoozeAlarm*, and 3) *AutoMonitor* and *ThresholdSentry*. When starting out, these sections should be completed in order as they build from the simplest application to the most complex. At the end of each section there is a Troubleshooting guide for commonly encountered problems. At the end of the chapter is a list of helpful commands that can be used from the BluSH prompt to aid with network configuration, testing and debugging.

3.1 Getting Started

To ensure success in the use of the software provided by the ISHMP Services Toolsuite, it is important that the Getting Started for Advanced Users and Developers guide has been successfully completed (http://shm.cs.uiuc.edu/documentation.html). Also, ensure that you have downloaded the most recent version of the ISHMP Services Toolsuite (http://shm.cs.uiuc.edu/software.html).

3.2 *RemoteSensing* without *SnoozeAlarm*

The *RemoteSensing* application allows the collection of synchronized sensor data from a network of Imote2s. Depending on the command that is given when the application is run, the data will either be nominally synchronized but not resampled or synchronized and resampled to eliminate any sample rate discrepancies between the sensor nodes or random delay in the start of sensing. For this example, the *SnoozeAlarm* application will

be disabled so that the leaf nodes are always awake. You will need the following components to install and use *RemoteSensing*:

- One Imote2 to act as the gateway node
- One USB cable
- At least one Imote2 with sensor board to act as the network (leaf node(s)).
- A battery board for each of the leaf nodes supplying at least the minimum voltage required by the sensor board being used (USB cable(s) may also be used to power the leaf nodes)
- Interface board (IIB2400) for gateway node communication with the PC

Note: Section 3.2 contains almost same information with ISHMP Getting Started Guide except that it uses TinyOS environment and ISHMP applications should be compiled and programmed on the motes. In this User's guide, `imote2comm` tool is used mostly while `autocomm` tool is used in ISHMP Getting Started Guide. Difference is that the `autocomm` creates output and log files automatically.

Step (1): Compiling and programming the motes

Open the Makefile (shm/tools/RemoteSensing/Makefile) in a text editor and ensure that the correct sensor board is selected (by typing the sensorboard's name right after `SENSORBOARD =`) and that *SnoozeAlarm* is disabled by including a "#" at the beginning of the corresponding line. The Makefile should look something like the text in Figure 6 (when using the ISM400, formerly SHM-A, sensor board).

```
COMPONENT = RemoteSensingC

# supported sensorboards: ITS400CA, ITS400CB, SHM_A, SHM_H, SHM_DAQ
SENSORBOARD = SHM_A

# configuration parameters, uncomment only if changing default
values
#UART_SPEED = UART_BAUD_921600
#SHMA_CUSTOM_FILTER_RATE = 1000
#RFPOWER = 31
#RFCHANNEL = 25

# optional components, uncomment to enable
USE_WATCHDOG = 1
#USE_CHARGER_CONTROL = 1
#USE_SNOOZE_ALARM = 1
#USE_MULTIHOP = 1

# required components
USE_SENSING_UNIT = 1

include $(SHMLIB)/ISHM/Makerules
```

This example has selected the MEMSIC ISM400 board (formerly the SHM-A)

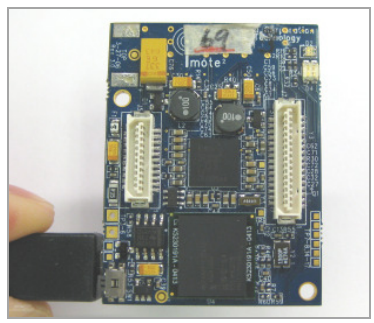
Figure 6. Makefile for *RemoteSensing* (shm/tools/RemoteSensing/Makefile).

Open a Cygwin window, go to the *RemoteSensing* directory (`cd $SHMROOT/tools/RemoteSensing`) and compile *RemoteSensing* (using the `make imote2`) as shown in Figure 7:

```
Cygwin
[mechitov@osl10 ~]$ cd $SHMROOT/tools/RemoteSensing
[mechitov@osl10 RemoteSensing]$ make imote2
```

Figure 7. Compiling *RemoteSensing*.

Once *RemoteSensing* has successfully compiled, install it via USB (using the `make imote2 reinstall` command as shown in Figure 8) on the gateway node (to the Imote2, not to the Interface board) and as many leaf nodes as you wish to test (up to 40).



a) Installation on the Imote2

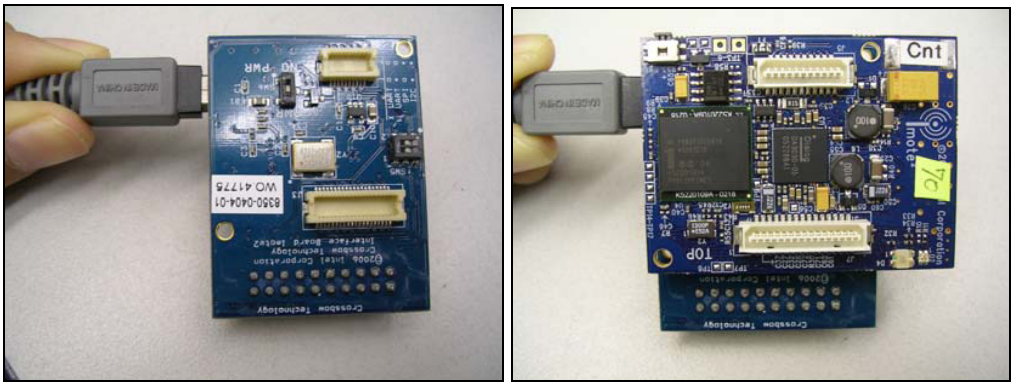
```
Cygwin
C:/cygwin/home/mechitov/shm/sensorboards/SHM_A/AccelSensorM.nc:
lkTxRx.BulkTxRx' called asynchronously from `getData'
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:11
.sendDone' called asynchronously from `sendFailed'
C:/cygwin/opt/tinyos-1.x/beta/platform/imote2/UARTBufferM.nc:19
iveData.receive' called asynchronously from `ByteComm.rxByteRea
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:97
omic accesses to shared variable `stateRadio':
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:34
-atomic read
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:35
-atomic write
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:36
-atomic read
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:36
-atomic write
C:/cygwin/opt/tinyos-1.x/tos/lib/CC2420Radio/CC2420RadioM.nc:59
-atomic read
    compiled RemoteSensing to build/imote2/main.exe
           535372 bytes in ROM
            38820 bytes in RAM
             9216 bytes in STACK
           214108 bytes available in HEAP
xscale-elf-objcopy --output-target=binary build/imote2/main.exe
[mechitov@osl10 RemoteSensing]$ make imote2 reinstall
```

b) Installing *RemoteSensing* (using the `make imote2 reinstall` command)

Figure 8. Installation of *RemoteSensing* on the gateway node.

Step (2): Setting up the network

Connect the gateway node to the Interface board (IIB2400) via the two connectors on the bottom of the Imote2. Connect the mini USB end of the USB cable into the gateway node interface board (not the Imote2, see Figure 9) and press the reset button on the Imote2 to turn on the power. Stack the leaf nodes with sensor boards on the battery boards and turn the power on (via slider switch on the side). All nodes (gateway and leaf nodes) should have solid red LEDs.



a) usb → Interface board(IIB) b) stack Imote2 on IIB

Figure 9. Gateway node composition.

RemoteSensing requires the use both of the UART serial ports that are provided by the IIB. One of the ports is used for sending commands receiving debug messages from the gateway node and the second port is to communicate data. Now you need to determine the port numbers for each of the new ports:

- Open the Windows Control Panel. Select "`System`"; choose the "`Hardware`" tab; click on "`Device Manager`". An example of "`Device Manager`" window is shown in Figure 10.
- Expand the "`Ports (COM & LPT)`" category. You should see something like:
  - `USB Serial Port (COM4)`
  - `USB Serial Port (COM5)`

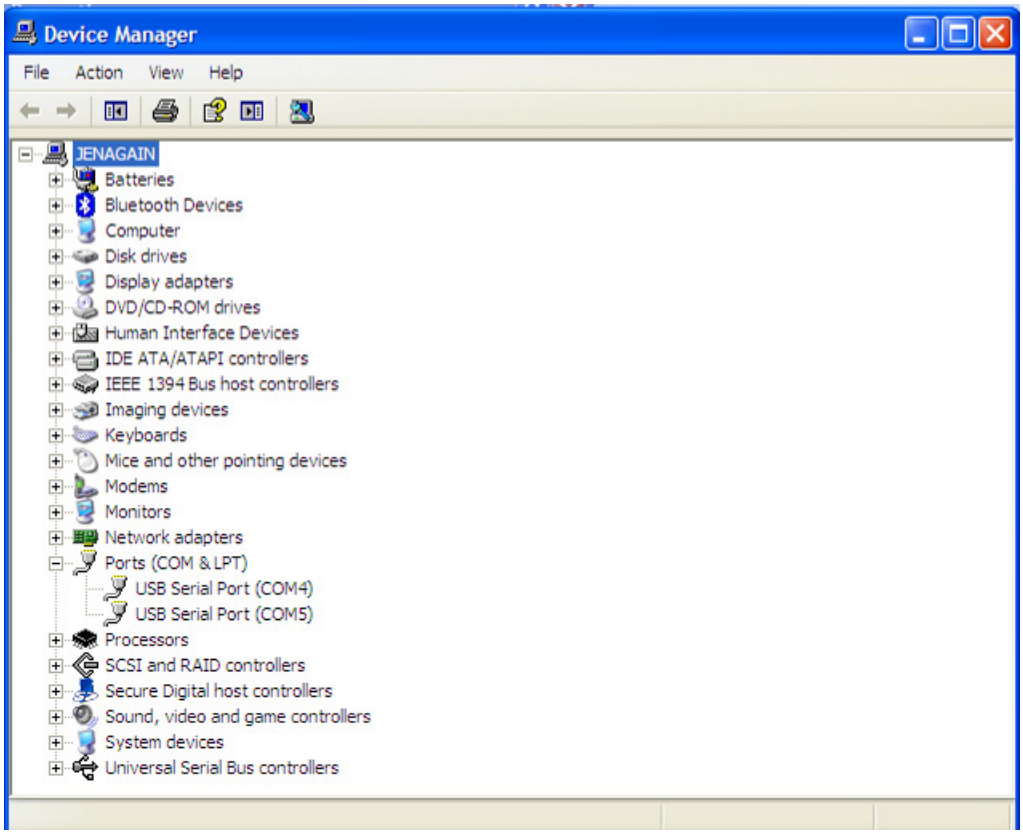


Figure 10. Device Manager listing the COM ports for the Interface board.

- If you cannot see the installed ports in Device Manager or access the COM ports after installing the drivers, reboot the computer.
- Write down the port values (COM4 and COM5 in this example). This guide will refer to these ports later as "`data UART`" for the lower port number and "`debug UART`" for the higher port number. You will need to know these identifiers to interact with the Imote2.

- **Note**: If you have multiple IIBs, each new board you connect to the PC will create its own set of COM ports so keep track of which COM ports belong to which board!

- **Note**: If either COM Port number is larger than COM16, the IIB interface board may not function properly. You can change the COM Port number to a value less than 16 by doing the following:
    - Right click on the "`USB Serial Port`" to open Properties as shown in Figure 11.
    - Open the "`Port Settings`" tab and click on "`Advanced`".
    - Choose a COM port number from 3 to 16 to assign to the UART port using the "COM Port Number" drop down box as shown in Figure 12. The "`(in`

use)" label can be ignored; you can choose these port numbers as well if between COM3 and COM16.

- Exit the "Device Manager" and then open it again to make sure that the changes have been made to your COM port numbers.

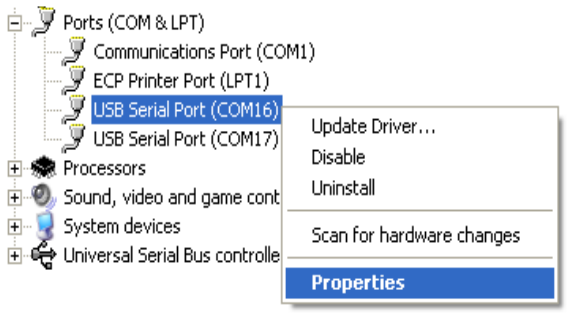


Figure 11. Opening the USB Serial Port Properties Menu.

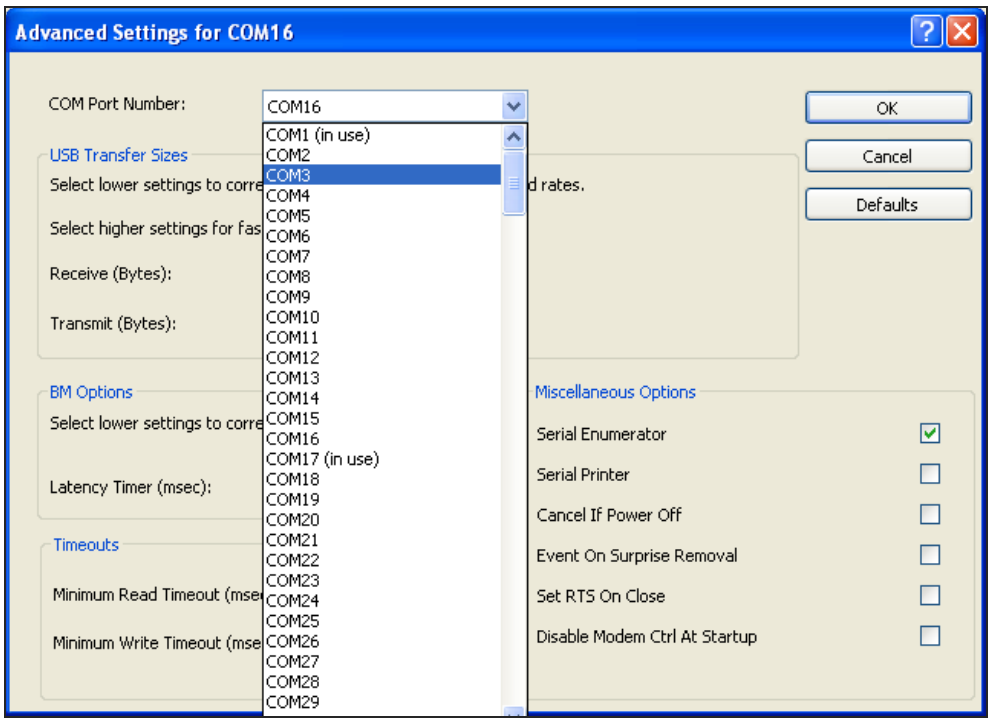


Figure 12. Changing the COM port number (if required).

- In the Cygwin window, run:

```
imote2comm -d COMy
```

where COMy is the higher COM port. Press <enter> a few times until a BluSH prompt appears. If a BluSH> prompt does not appear, check if the correct COM port has been entered. This step is shown in Figure 13.

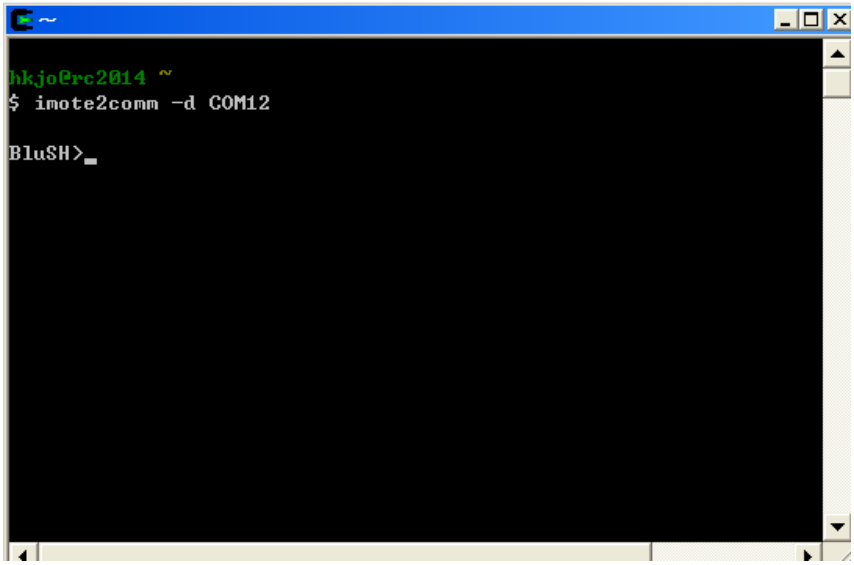


Figure 13. Entering the BluSH environment using imote2comm.

To see a list of the commands available on the Imote2, use the BluSH command `ls`. Some of the available commands are directly used by interact with the *RemoteSensing* application, while others are utilities used in other applications available in the ISHMP Services Toolsuite. Figure 14 shows the commands available from the preinstalled ISHMP Services Toolsuite. See Getting Started Guide for New Users (Appendix B) at http://shm.cs.uiuc.edu/NewUser.html for detailed information about these commands.

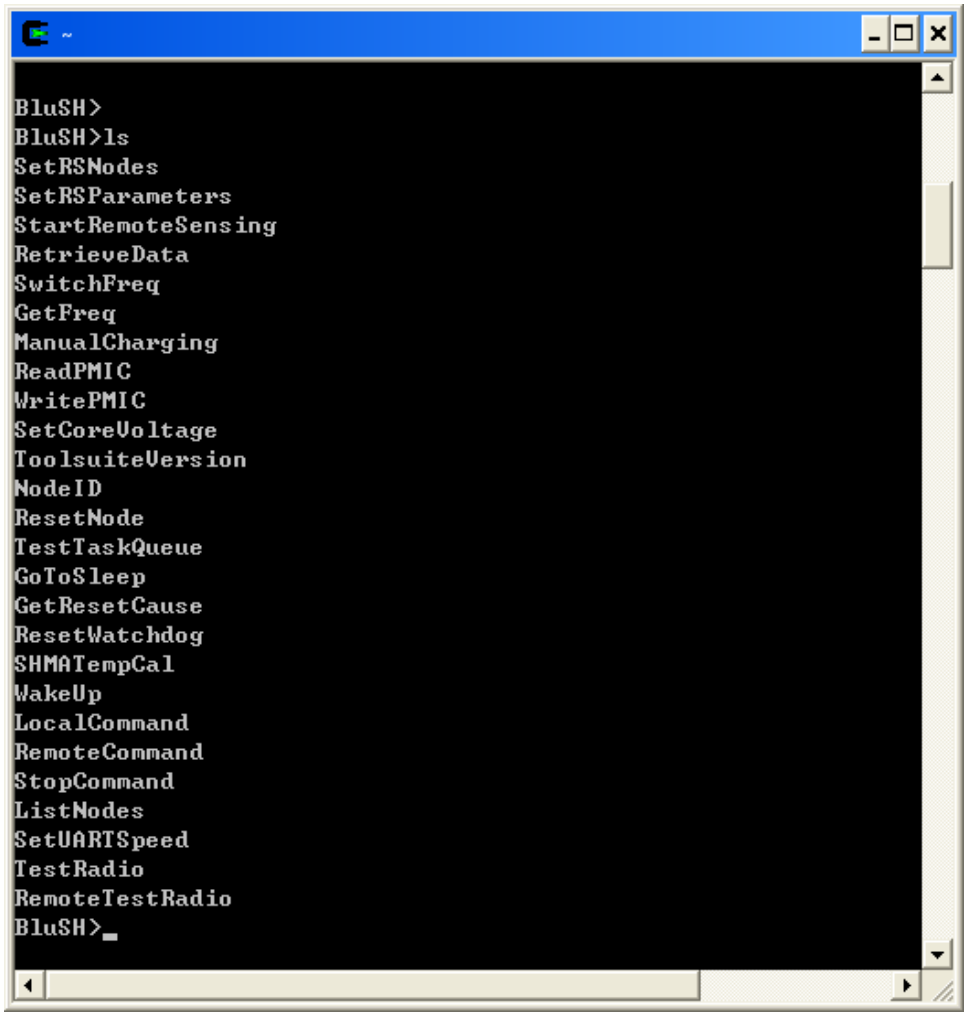


Figure 14. Available BluSH commands.

- Before collecting any data from your network, you will need to know the node IDs, in decimal base, of the Imote2s that comprise your network. To find the node ID, look at the string of letters and numbers on the back of the Imote2 below the barcode and record the last two characters. These two characters are the node ID in hexadecimal. In Figure 15, for example, the hexadecimal node ID is 78. You can convert this base-16 number to base-10 using the Windows Calculator application. As shown in Figure 16, first click 'Hex' and type hexadecimal node ID (78) and then click 'Dec' to get decimal node ID (i.e., 120 in this case).

Figure 15. Imote2 barcode showing the Node ID (78) in hexadecimal.

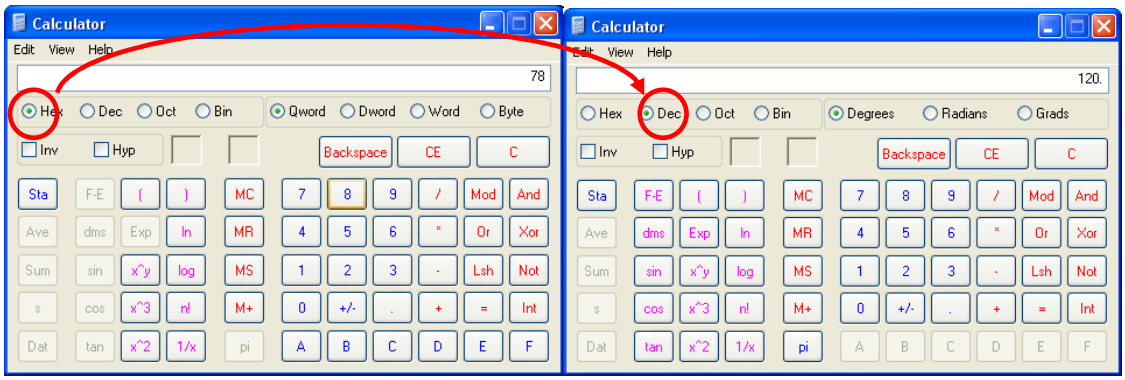


Figure 16. Converting hexadecimal to decimal using the Windows calculator.

- The sampling parameters used in *RemoteSensing* are user specified and entered as parameters when the function is called from the BluSH. You can set the channels to be sampled, the number of data points acquired per channel, and the sampling rate.

- To save the collected data to a file, you will need to open a second DOS command window. In the new command window, use the `cd` command to change the directory to the location where the data will be saved and then type:

  - `imote2comm -n -o out.txt COMx`

  where COMx is the lower COM port. The "-n" flag will cause the data to be written only to the output file and not be shown on the screen. Using this flag minimizes "printing" errors when writing to the saved file.

Step (3): Running *RemoteSensing*

The application *RemoteSensing* will collect synchronized data from the leaf nodes, transfer it back to the gateway node, and write the output to the file specified in the second command window created in Step (2).

- Before issuing any commands, make sure your leaf nodes are ready to collect data by checking that the sensor boards are firmly attached, the antenna is tightened, the nodes are turned on, and the red LEDs are shining.
- To collect synchronized data, go to the first command window and type the following command at the `BluSH>` prompt:
  - `SetRSNode <nodeId> [nodeId] [nodeId] …`

  where you list the decimal value of the node IDs of all the leaf nodes (see Figure 17).

```
BluSH>
BluSH>SetRSNodes
- usage: SetRSNodes <nodeId> [nodeId] [nodeId] ...
BluSH>
BluSH>
BluSH>
BluSH>SetRSNodes 25
- RemoteSensing nodes set.
BluSH>
```

Figure 17. Starting the *RemoteSensing* application.

- Next, input the sensing parameters as shown in Figure 18:
  - `SetRSParameters <channelMask> <numSamples> <samplingFrequency> <syncSensing>`

  The `SetRSParameters` arguments are:
  - `channelMask`: this parameter designates the combination of sensor channels that will be collected, *e.g.*:
    - 1 = channel 1
    - 13 = channels 1 and 3
    - 123 = channels 1, 2 and 3
  - `numSamples`: number of request data samples
  - `samplingFrequency`: sampling rate in Hz (see Table 2 and Table 3 for supported rates)
  - `syncSensing`: user can select whether to perform time synchronization before sensing or not
    - `1` = Sensing after network time synchronization (Recommended when you have multiple leaf nodes)
    - `0` = Sensing without synchronization (Recommended when you have single leaf node)
- The application is now configured; you can initiate data collection in the *RemoteSensing* application by typing:
  - `StartRemoteSensing <clearmetaData>`

- 1 = First metadata block will be erased such that data will be stored in the first block.
- 0 = Data will be stored in the next available block.

- When synchronized sensing is selected, gateway node will begin to synchronize the network at this point. Once network synchronization is complete, data collection will begin and the LEDs on all of the nodes will turn green. When the leaf nodes are finished sensing, their LEDs will turn off. Wait until some debug logs appears.

- When sensing is done, you can now collect the latest data by typing `RetrieveData -1` (see Figure 18):
  - RetrieveData -1 `<nodeId> [nodeId] [nodeId] …`

  If multiple data sets are stored in flash memory, they can be retrieved by replacing '-1' to a data set index (1 through 9). Note that the maximum number of data sets can be stored in flash memory is nine; if the tenth data set is collected with clearmetaData = 0, it will overwrite the first data set.

- When all the leaf nodes have successfully sent their data to the gateway node and the data has been written to the PC, the command window will return the message "Finished writing output." Figure 18 shows an example of the commands issued and the debug output that you will see in the debug UART command window.

- Note that Date and time appear in debug log window will be different from your Base Station PC time.

- Synchronizing base station computer time and the gate way *(Recommended)*
  - Collected data sets are stored persistently in flash memory on the leaf nodes. A leaf node can hold up to 9 sets of data block stored successively.
  - `Retrieve -1 <nodeId>…` retrieves the latest data set, but the user also can retrieve older data sets by synchronizing the base station computer and the gateway (See Figure 19)
  - Also, when a node fails to collect data in a network, and the gateway attempt to retrieve all the data set in a network, data may not be the synchronized flash data set. User can tell if the retrieved data sets are actually synchronized, by timestamp in output file.
    Note: This must be done prior to collecting the data sets with the *RemoteSensing* application.
  - Open a new Cygwin window (keep "`autocomm -d COMy`" opened in earlier steps)
  - Run `forwarder`

        - Debug logs "Forwarder started on port … Waiting for connections…" will appear
    - Open another Cygwin window
    - Run "autocomm -s LocalHost Comx"
        - Debug log "Synchoronizing time with imote2" will appear
        - "Synchronized time…"will appear in Debug window connected earlier with "autocomm -d COMy".
        - At the same time, a debug message "New client connection established." will appear in the forwarder Cygwin window.

```
BluSH>
BluSH>SetRSNodes 25
- RemoteSensing nodes set.
BluSH>SetRSParameters 123 1000 100 1
- RemoteSensing parameters set.
BluSH>
BluSH>StartRemoteSensing 1
- Erasing flash memory of remote nodes ...
BluSH>
- Erase flash command sent to nodes 25

- Flash memory erased.
- 1 nodes are set.
- parameters are set

- starting time synchronization, wait 30 seconds...
- finished Time Sync.

- starting data acquisition in 21 seconds...
- sensing parameters sent to node 25
- sensing started. waiting 14 seconds...
- resampling data, waiting 7 seconds...
- sensing finished

- Sensing in remote nodes finished.
- Responsive nodes are 25

BluSH>
BluSH>RetrieveData -1 25
- Requesting the most recent data from node 25...
BluSH>- Flash data received, unpacking ...
- Finished receiving data from node 25.  Writing output...
- timestamp = 3970432231
- Date and time: 1970-01-01 00:20:21
- Finished writing output.
```

Figure 18. Output from the *RemoteSensing* application.

```
Robin@        ~
$ autocomm -d COM4

BluSH>_
```

```
Robin@        ~
$ forwarder
Forwarder started on port 2010.  Waiting for connections...
```

```
Robin@        ~
$ autocomm -s LocalHost COM3
Synchronizing time with Imote2.
_
```

```
Robin@        ~
$ autocomm -d COM4

BluSH>Synchronizing time with PC.  Local time now: 2011-01-10 09:22:47.
_
```

Figure 19. Debug messages for time synchronizing the Gateway with Base station PC.

Now that you have successfully run the *RemoteSensing* application, you can view the collected data in the program of your choice.

- Press <Ctrl-C> in both command windows to close the `autocomm` application.
    - The output in "out.txt" will contain the index and sensor data in a tab-delimited format that is suitable for importing into either Excel or Matlab.
- As shown in Figure 20, the columns in the output file are:
    - *node* – the Imote2 node ID
    - *ch. n* – sensor data for accelerometer channel *n*

Date and Time:	1970-01-01 00:07:51		
node	ch.1	ch.2	ch.3
25	13774	13960	20683
25	13772	13963	20682
25	13772	13964	20685
25	13773	13963	20687
25	13773	13961	20686
25	13771	13959	20686
25	13770	13961	20686
25	13770	13961	20683
25	13771	13959	20680
25	13773	13957	20682
25	13773	13957	20688
25	13772	13959	20691
25	13771	13960	20690
25	13771	13957	20688
25	13773	13956	20689
25	13774	13958	20688
25	13774	13961	20686
25	13773	13960	20685
25	13772	13960	20686
25	13771	13961	20686
25	13770	13962	20683
25	13770	13960	20681
25	13771	13958	20682
25	13771	13957	20687
25	13770	13958	20680

Figure 20. Sample output of *RemoteSensing* application (with resampling).

Note: You can collect the raw data that has not been resampled using the fourth input of Set`RSParameter, 0`, but is still approximately synchronized.

Troubleshooting Tips for *RemoteSensing* without *SnoozeAlarm*

1. **Problem:** After uploading the application image, the mote continuously reboots (the LED flashes briefly about once every 2 seconds).

   Solution: This occurs when incorrect Flash constants are loaded on the mote. It is necessary to reprogram the mote with a fresh set of Flash constants:

   ```
   cd $SHMROOT/tools/WriteFlashConstants
   make imote2 install
   ```

2. **Problem:** A node is repeatedly exhibiting unexpected behavior and the battery voltage is adequate.

   Solution: This could be caused by a hardware problem. Try to isolate the source of the hardware problem by switching the battery board and sensor board.

3. **Problem:** Printing out data fails with messages like "ReliableSerial: send failed".

   Solution: You may need to reduce the port speed. Run the following two commands in the Imote2's BluSH shell and on the PC, respectively:

   ```
   SetUARTSpeed 460800
   autocomm -b 460800 -n -o out.txt COMx
   ```

3.3 *RemoteSensing* with *SnoozeAlarm*

When *SnoozeAlarm* is enabled on the leaf nodes it puts them in a continuous sleep/wake cycle where the node is in a deep sleep mode most of the time but periodically wakes to listen for commands. When the leaf node receives a command it remains awake until put back to sleep or reset. The purpose of *SnoozeAlarm* is power savings as the node uses less than 10 percent of the power when it is in the deep sleep mode than when it is in an idle awake mode.

Step (1): Compiling and programming the remote motes

Since *SnoozeAlarm* is only enabled on the leaf nodes, you will do two separate compilations of the software, one for the gateway node and one for the leaf nodes. The gateway node will be compiled and installed exactly as in the previous example (see

Step (1) in *RemoteSensing* without *SnoozeAlarm*). Before compiling the software for the leaf nodes, open the Makefile (shm/tools/RemoteSensing/Makefile) in a text editor and ensure that the correct sensor board is selected. Enable *SnoozeAlarm* by uncommenting the corresponding line (remove the "#" from the beginning of the line). The Makefile should look something like the following (when using the ISM400, formerly SHM-A, sensor board):

```
COMPONENT = RemoteSensingC

# supported sensorboards: ITS400CA, ITS400CB, SHM_A, SHM_H, SHM_DAQ
SENSORBOARD = SHM_A

# configuration parameters, uncomment only if changing default
values
#UART_SPEED = UART_BAUD_921600
#SHMA_CUSTOM_FILTER_RATE = 1000
#RFPOWER = 31
#RFCHANNEL = 25

# optional components, uncomment to enable
USE_WATCHDOG = 1
#USE_CHARGER_CONTROL = 1
SNOOZE_ALARM_REMOTE = 1
#ENABLE_CHARGER_CONTROL = 1
#USE_MULTIHOP = 1

include $(SHMLIB)/ISHM/Makerules
```

This example has selected the MEMSIC ISM400 board (formerly the SHM-A)

Figure 21. Enabling *SnoozeAlarm* in the Makefile for *RemoteSensing*.

Save the updated Makefile then compile *RemoteSensing* with *SnoozeAlarm* enabled and install it (via USB) on the LEAF NODES ONLY. In the case of a node with *SnoozeAlarm* already enabled, special steps must be taken during the USB installation to work around the periods of deep sleep (during which installation cannot occur). The following steps allow the USB installation for nodes in *SnoozeAlarm* mode:

1. Remove Imote2 from the battery board
2. Plug the USB cable into the Imote2 but do not press the reset button
3. In the command line type "`make imote2 reinstall`" but do not press <enter>
4. Press the reset button on the Imote2
5. Immediately hit <enter> on the command line
6. Wait until you get a "`Device detected`"
7. Unplug and replug the USB cable into the Imote2
8. Press the reset button on the Imote2
9. The image should now low but if it doesn't unplug the Imote2 and start from step one.

When a *SnoozeAlarm*-enabled node is powered (or the reset button is pressed) the node will remain awake for WAKE_INTERVAL ms. After this period, the node will sleep for SLEEP_INTERVAL seconds. The default times (WAKE_INTERVAL = 10 ms and

SLEEP_INTERVAL = 10 seconds) are defined in shm/lib/ISHM/FlashConstants.h. If different values are desired, they must be changed prior to compilation and installation as they cannot be changed afterwards.

Step (2): Waking the network

The leaf nodes in *SnoozeAlarm* mode must be woken up before the motes can be used normally. Any remote command invocation (for example, `RemoteCommand Vbat` or *RemoteSensing*'s `StartRemoteSensing`) will automatically wake up the remote nodes. This can also be done manually with the `WakeUp` command:

```
WakeUp <nodeId> [nodeId] [nodeId] ...
```

In the case of the `WakeUp` command, wait for: "`All nodes awake`" OR a list of the nodes that were successfully woken up before the request times out. The leaf nodes should have solid red LEDs once they receive the command to wake up.

Step (3): Running *RemoteSensing*

Continue with the *RemoteSensing* commands as outlined in the previous section. Only successfully woken/responsive nodes will participate. Once each node finishes sending its data to the gateway node it will put itself back in *SnoozeAlarm* mode.

Troubleshooting Tips for *SnoozeAlarm*

1. **Problem:** The `WakeUp` command consistently times out before waking all of the requested nodes.

   Solution: This may be due to a poor communication environment. One option is to increase the additional wake up time per node in the wakeup time out by increasing NODE_WAKE_UP_TIME (in ms) in FlashConstants.h. Another option is to increase the WAKE_TIME (in ms) in FlashConstants.h.

3.4 *AutoMonitor* and *ThresholdSentry*

ThresholdSentry is an application that periodically wakes one leaf node at a time within a pre-defined network of sentry nodes to measure data for a short period of time. The peak of the normalized measured data is compared against a specified threshold level. If the threshold is exceeded on the sentry node, it sends a flag back to the gateway node. If the threshold is not exceeded, the sentry node is put back to sleep (if used with *SnoozeAlarm*) or is reset. The process is repeated at a set interval for each of the sentry nodes.

AutoMonitor is an application that runs on the gateway node to allow autonomous network operation combining *ThresholdSentry, RemoteSensing* and *AutoUtilCommand* with automatic data file and debug log generation. When *AutoMonitor* receives the flag that the threshold has been exceeded on a sentry node, it attempts to wake up the whole network and initiate *RemoteSensing*. The user can define a maximum number of *RemoteSensing* events that occur in a defined period of time to eliminate excessive network wake ups.

In addition to managing the operations of the network in response to *ThresholdSentry*, *AutoMonitor* is also responsible for creating data files on the PC when *RemoteSensing* completes and for periodically creating log files of the debug output. The log files can be used by the network administrator to assess if there are any problems with the network or specific nodes.

When *AutoUtilCommand* is executed, selected nodes send the reading value of its voltage level, temperature, charge status and light to the gateway node. Debug logs will appear for selected leaf nodes send. A text file will be generated containing final reading value of voltage, temperature, charge status and light after *AutoUtilCommand* is done. The user can define interval time to execute *AutoUtilCommand.*

Upon initiation of *AutoMonitor* via an input file, the network requires minimal interaction from the user, except to monitor the output files to ensure things are behaving correctly.

The SHM Services Toolsuite supports the *multiple threshold levels* and *WindThresholdSentry* as of version 3.0. The multiple threshold levels, having separate maximum number of events, make it possible to measure both rare but strong responses and frequent but low-level ambient vibration during a given period, which enables more effective monitoring. In addition to acceleration based *ThresholdSentry*, wind based *WindThresholdSentry* is available with an anemometer & SHM-W board. Detail information about the *WindThresholdSentry* will be addressed in near future. In this stage, these can be ignored safely.

Step (1): Compile and install

AutoMonitor is installed on the *GATEWAY NODE* ONLY. The leaf nodes are installed with *RemoteSensing* (with or without *SnoozeAlarm* enabled). See the *RemoteSensing* sections for instructions on compiling and installing *RemoteSensing* on the leaf nodes. To compile and install *AutoMonitor* for the gateway node open a Cygwin window, change the directory to *AutoMonitor* and compile as shown in Figure 22.

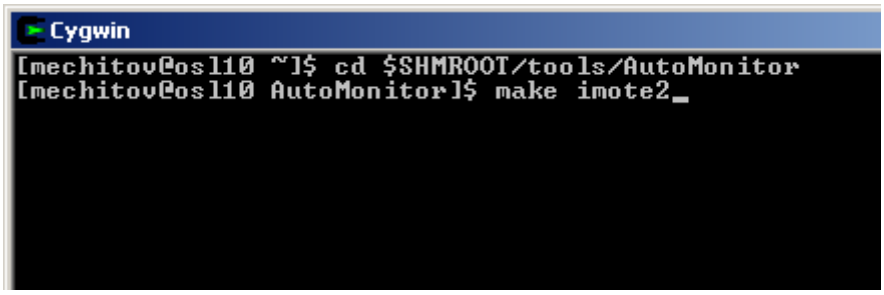


Figure 22. Compiling *AutoMonitor*.

Install on the gateway node via `make imote2 reinstall`. The LED will be solid red upon installation.

Step (2): Create an input file

The information contained in the input file serves the following purposes:

1. Define the *RemoteSensing* network nodes
2. Define the *RemoteSensing* sensing parameters
3. Define the *ThresholdSentry* sentry network
4. Define the *ThresholdSentry* parameters
5. Define the maximum number of allowed network sensing events within a given time period
6. Define the refresh time period for generating debug logs and resetting the *RemoteSensing* event count
7. Define the Autocommand Utility network nodes
8. Define the Autocommand parameters
9. Start the *ThresholdSentry* component

Figure 23 shows an example *AutoMonitor* input file:

```
INPT 14
RefreshPeriod 2
RSSetup 119 65
GDSetup 123 3000 280 1
SentrySetup 123 100 20 30 5 10 500 3 1 1
WindSentrySetup 2 1 4 1
WindSentryNode
ReadVbat 119 65
ReadTemp 119 65
ChargeStatus 119 65
ReadLight 119 65
AutoCommandSetup 1 10 20 2
THSentryStart 119 65
DDASensorTopology 119 65
DDANExTSetup 1024 5 512 3 1 1 100 50 3 13 0 1
```

Figure 23. Sample input file for AutoMonitor application.

The input lines are:

- INPT <numlines>
  - MUST BE THE FIRST LINE
  - numlines: indicates the number of commands (lines) in the file, not including the INPT line itself.

- RefreshPeriod <period (hours) >
  - Defines the time (hours) between resetting the count for sensing events (if ThresholdSentry exsits) and creating a debug output text file (if using autocomm with the -d and -o flags).

- RSSetup <nodeId> [nodeId] [nodeId] ...
  - Defines the network that will participate in network-wide
  - RemoteSensing

- GDSetup <channelMask> <numSamples> <samplingRate> <syncSensing>
  - Defines the sensing parameters when *RemoteSensing* is triggered

- SentrySetup <channelMask> <samplingRate> <checkTime> <checkInterval> <threshold1> <rsmax1> <threshold2> <rsmax2> <period> <remotesensing=1 or DDA=2> <priority>
  - Defines the *ThresholdSentry* parameters:
    - channelMask: the channels used on the sentry node when checking the response levels (see the description on how to input the channelMask in the instructions for *RemoteSensing*)
    - samplingRate: the sampling rate used on the sentry node when checking the response levels (ensure this is a supported sampling rate for the sensor boards being used)

- checkTime: the amount of time (seconds)the sentry nodes senses when checking the response levels
- checkInterval: amount of time (seconds) between sentry node checks
- threshold1(smaller value): the value against which the sentry data is checked (mg). If the max sensed value > threshold and the maximum number of *RemoteSensing* events has not already been exceeded, the network is woken to perform network-wide sensing
- rsmax1: the maximum number of network-wide sensing events for threshold1 allowed in the time period specified by <period>.
- Threshold2(larger value): 2nd threshold acceleration
- Rsmax2: the maximum number of network-wide sensing events for threshold2. If no need, enter 0.
- period: the time (hours) between resetting the count for sensing events and creating a debug output text file (if using autocomm with the -d and -o flags).
- Either running RemoteSensing or DDA when the network of sensors is awake. For RemoteSensing, use 1.
- priority: priority of Thresholdsentry applicatoin, bigger number means higher priority

- WindSentrySetup <threshold_w> <rsmax_w> <threshold_w2> <rsmax_w2>
  - Defines the *WindThresholdSentry* parameters:
    - threshold_w(smaller value): threshold wind speed(m/s)
    - rsmax_w: max number of events
    - threshold_w2(larger value): 2nd threshold wind speed(m/s)
    - rsmax_w2: 2nd max number of events (if no need, enter 0)

- WindSentryNode <nodeID> [nodeID] [nodeID] ...
  - Define wind-sentry node id **(if no wind-sentry, just left empty)**

- THSentryStart <nodeId> [nodeId] [nodeId] ...
  - Defines the sentry nodes used in *ThresholdSentry*

- ReadVbat <nodeID> [nodeID] [nodeID] ...
  - Define Battery Voltage node id (if no Battery Voltage node, just left empty)

- ReadTemp <nodeID> [nodeID] [nodeID] ...
  - Define Temperature node id (if no Temperature node, just left empty)

- ChargeStatus <nodeID> [nodeID] [nodeID] ...
  - Define Charge Status node id (if no Charge Status node, just left empty)

- ReadLight <nodeID> [nodeID] [nodeID] ...

- Define Light node id (if no Light node, just left empty)

- AutoCommandSetup <firstExecutinDelay> <checkInterval> <maxTimes> <priority>
  - firstExecutionDelay: the delay between first AutoUtilCommand execution and start of AutoMonitor
  - checkInterval: amount of time (miniutes) between AutoUtilCommand checks
  - maxTimes: the maximum number of checks in the time period specified by <RefreshPeriod>
  - priority: priority of AutoUtilCommand application, bigger number means higher priority

DDASetup / DDATopology : This is for Decentralized Data Aggregation(DDA) application. Detail usage will be added in near future. In this stage, these lines can be safely ignored.

Save the input file in an easily accessible location (eg. shm/tools/AutoMonitor/input.txt)

Step (3): Start *AutoMonitor*

Turn on all nodes that make up the network and ensure they correspond to the node Ids in the input file. If *SnoozeAlarm* is enabled on the leaf nodes they will wake periodically for a short period of time during which their LEDs will flash red. Attach the gateway node to the interface board and insert the mini USB connector into the interface board (not the Imote2). Plug the other end of the USB cable into the PC and press the reset button on the Imote2 to power it on; the LED should be solid red. Determine the COM ports associated with the interface board by going to My Computer → View System Information → Hardware → Device Manager.

Open two Cygwin windows, one for the debug UART and the second for the input/output UART. In the first Cygwin window run:

```
autocomm -d -o debug.txt COMy
```

The autocomm application differs from imote2comm in that it allows the automatic generation of data and debug files. The file (debug.txt) is opened when the above line is run. It will be closed and saved after the specified *AutoMonitor* refresh period of time expires after the application is started and will contain all of the debug output that is generated during that period of time. The file name will reflect the point in time it was closed, i.e. for a file that is generated at on June 12, 2009 at 9:08:05 am the file name will be:

```
debug_20090612-090805.txt
```

Press <enter> a couple of times, until you get the BluSH prompt. Then, at the BluSH prompt type:

```
StartAutoMonitor n
```

where `n` is the number of times the `WakeUp` command is attempted. In some cases, running the Wakeup command two times in a row ensures more nodes wake up prior to moving on to the next step. You will then be prompted for the input file. In the next Cygwin window run the following two lines send the input file and setup the output file for the collected data:

```
autocomm –i input.txt COMx
autocomm –n –o output.txt COMx
```

and press <enter>. The application should start automatically as indicated in the first window. A data file will be generated each time *RemoteSensing* is run. The name of the file will correspond with the time it was generated. For example, a data output file generated on June 12, 2009 at 1:10:36 pm will be named:

```
output_20090612-131036.txt
```

Once this above command line is run, the timer starts for the *ThresholdSentry* and *Autocommnad* check interval specified in the input file. When the check interval time for *ThresholdSenty* expires, the gateway node will send a sentry request to the first node in the list of sentry nodes. It may take several seconds to wake the sentry node at which point its LED will turn green and it will start sensing for a short period of time. If the threshold is not exceeded on the sentry node it will go back into the *SnoozeAlarm* mode and the debug output will display "Threshold not exceeded on node X". The process will start again with the next sentry node and will continue until the threshold is exceeded. If the threshold is exceeded on a sentry node it will alert the gateway node and remain awake while it awaits the next command. The gateway node will try to wake up the entire network. Once it either successfully wakes up the entire network or times out before all nodes are awake, it will initiate *RemoteSensing* with the successfully woken nodes. The debug output will be similar to the manual operation of *RemoteSensing*. Each leaf node will put itself back in *SnoozeAlarm* mode once its data is reported. At the conclusion of printing data, an output file will be generated and according to interval time, either *AutoUtliCommnad* will be executed or *ThresholdSentry* will be restarted where it left off (with the next sentry node in the sequence).

When the check interval time for *AutoUtilcommand* expires, the selected leaf nodes will return reading value for voltage, temperature, charge status and light. Temperature and light check will be repeated 6 and 3 times respectively. After *AutoUtilCommand* is done, a text file will be generated containing final reading value of voltage, temperature, charge status and light. If a node's voltage level is lower than the minimum required battery voltage for measuring acceleration, 3.6 V (see section 2.1), the node will be

execluded from the *AutoMonitor* network. Also, when *ThresholdSentry* and *AutoUtilCommand* execution fall at the same time, the event with the higher priority will be executed first, and lower priority will be followed.

The Figure 24shows the debug input/output for *AutoMonitor*. Note that even a node is not responsive the application will be continued to run without it.

```
BluSH>StartAutoMonitor 1
AutoMonitor started, ready for input...
BluSH>rec: RefreshPeriod 2
- Flush AutoMoitor every 2.00 hour(s).
rec: RSSetup 25 151 65
rec: GDSetup 123 3000 280 1
rec: SentrySetup 123 100 20 30 100 10 500 3 1 1
        Priority = 1
rec: WindSentrySetup 2 1 4 1
rec: WindSentryNode
No wind sentry node ...
rec: ReadVbat 25 151 65
- RemoteVbat nodes set.
25 151 65
rec: ReadTemp 25 151 65
- RemoteTemp nodes set.
25 151 65
rec: ChargeStatus 25 151 65
- RemoteChargeStatus nodes set.
25 151 65
rec: ReadLight 25 151 65
- RemoteLight nodes set.
25 151 65
rec: AutoCommandSetup 1 10 20 2
Autonous UtilityCommand Setup with:
         Start time = after 1 min
         Number of checking = 20 times
         Checking interval = every 10 min
         Priority = 2
rec: THSentryStart 25 151 65
Starting sentry mode on 3 nodes: 25 151 65
ThresholdSentry setup with:
        Threshold = 100
        Time between checks = 30 sec
        Check time = 20 sec
Sentry network setup with 3 nodes: 25 151 65
rec: DDASensorTopology 151 65
rec: DDANExTSetup 1024 5 512 3 1 1 100 50 3 13 0 1
- Executing application "ThresholdSentry"...
Successfully woke up 1 node(s): 25

Sentry request sent to node 25.
Threshold not exceeded on node 25.
- Application "ThresholdSentry" is done!

- Executing application "ThresholdSentry"...
Successfully woke up 1 node(s): 151

Sentry request sent to node 151.
AccThreshold exceeded on node 151 at ch3, peak = 1352 mg
AccThreshold 2nd level exceeded, waking network to initiate network sensin
Successfully woke up 2 node(s): 25 151
Synchronized network sensing initiated with responsive nodes.
- RemoteSensing nodes set.
- starting data acquisition in 21 seconds...
- sensing parameters sent to node 25 151..
- sensing started. waiting 13 seconds...
- resampling data, waiting 8 seconds...151
- sensing finished
- Flash memory erased.
- Sensing in remote nodes finished.
- Responsive nodes are 25 151
- Requesting flash data from node 25 Set # 1...
```

Debug output created

Input file read

ThresholdSentry running

Threshold exceeded, waking network

RemoteSensing

```
- Requesting flash data from node 25 Set # 1...
- Flash data received, unpacking ...
- Finished receiving data from node 25.  Writing output...
- timestamp = 1532291768
- Date and time: 1970-01-01 00:07:51
- Requesting flash data from node 151 Set # 1...
- Flash data received, unpacking ...
- Finished receiving data from node 151.  Writing output...
- Finished writing output.

Data collected successfully!
The number of Acc triggering events occured, runcount = 0, runcount2 = 1
- Application "ThresholdSentry" is done!

- Executing onhold application "AutoCommand"
- Executing application "AutoCommand"...
Successfully woke up 2 node(s): 25 151
2 nodes awake: 25 151
TempNode: 2 nodes awake 25 151
ChargeNode: 2 nodes awake 25 151
LightNode: 2 nodes awake 25 151
- RemoteVbat nodes set.
25 151
- RemoteTemp nodes set.
25 151
- RemoteChargeStatus nodes set.
25 151
- RemoteLight nodes set.
25 151
```

AutoUtilCommand executed

```
Trying AutoCommand1 for node 25.
Remote command "Vbat" successfully sent to 1 of 1 node(s): 25
Remote command "Vbat" executed.
Battery voltage: 4.109V

Trying AutoCommand1 for node 151.
Remote command "Vbat" successfully sent to 1 of 1 node(s): 151
Remote command "Vbat" executed.
Battery voltage: 4.047V
AutoCommand1 finished for all 2 nodes.
Perform next AutoCommand

Trying AutoCommand2 for node 25.
Remote command "ReadTemp" successfully sent to 1 of 1 node(s): 25
Remote command "ReadTemp" executed.
Temperature (from Sensirion): 615.7 C

Trying AutoCommand2 for node 25.
Remote command "ReadTemp" successfully sent to 1 of 1 node(s): 25
Remote command "ReadTemp" executed.
Temperature (from Sensirion): 31.1 C

Trying AutoCommand2 for node 25.
Remote command "ReadTemp" successfully sent to 1 of 1 node(s): 25
Remote command "ReadTemp" executed.
Temperature (from Sensirion): 28.9 C

Trying AutoCommand2 for node 25.
Remote command "ReadTemp" successfully sent to 1 of 1 node(s): 25
Remote command "ReadTemp" executed.
Temperature (from Sensirion): 28.5 C
```

AutoUtilCommand tunning

```
Trying AutoCommand2 for node 151.
Remote command "ReadTemp" successfully sent to 1 of 1 node(s): 151
Remote command "ReadTemp" executed.
Temperature (from Sensirion): 28.4 C

Trying AutoCommand2 for node 151.
Remote command "ReadTemp" successfully sent to 1 of 1 node(s): 151
Remote command "ReadTemp" executed.
Temperature (from Sensirion): 28.3 C
AutoCommand2 finished for all 2 nodes.
Perform next AutoCommand

Trying AutoCommand3 for node 25.
Remote command "ChargeStatus" successfully sent to 1 of 1 node(s): 25
Remote command "ChargeStatus" executed.
Charging status: vBat = 2.650V vChg = 15.835V iChg = 0.000A chargeControl = 0

Trying AutoCommand3 for node 151.
Remote command "ChargeStatus" successfully sent to 1 of 1 node(s): 151
Remote command "ChargeStatus" executed.
Charging status: vBat = 2.650V vChg = 15.835V iChg = 0.000A chargeControl = 0
AutoCommand3 finished for all 2 nodes.
Perform next AutoCommand

Trying AutoCommand4 for node 25.
Remote command "ReadLight" successfully sent to 1 of 1 node(s): 25
Remote command "ReadLight" executed.
Illumination:      0 Lux

Trying AutoCommand4 for node 25.
Remote command "ReadLight" successfully sent to 1 of 1 node(s): 25
Remote command "ReadLight" executed.
Illumination:   1351 Lux

Trying AutoCommand4 for node 25.
Remote command "ReadLight" successfully sent to 1 of 1 node(s): 25
Remote command "ReadLight" executed.
Illumination:   1350 Lux

Trying AutoCommand4 for node 151.
Remote command "ReadLight" successfully sent to 1 of 1 node(s): 151
Remote command "ReadLight" executed.
Illumination:      0 Lux

Trying AutoCommand4 for node 151.
Remote command "ReadLight" successfully sent to 1 of 1 node(s): 151
Remote command "ReadLight" executed.
Illumination:   1191 Lux

Trying AutoCommand4 for node 151.
Remote command "ReadLight" successfully sent to 1 of 1 node(s): 151
Remote command "ReadLight" executed.
Illumination:   1191 Lux
- 4 types of utility commands finished.
- Printing ...
- Printing finished.
Remote command "Reset" successfully sent to 2 of 2 node(s): 25 151
- Application "AutoCommand" is done!
```

Printing output for *AutoUtilCommand*

Figure 24. Example of debug input/output for *AutoMonitor* application.

AutoMonitor, *ThresholdSentry* and *RemoteSensing* have some safeguard/power-saving features built in to ensure continuous operation. These features include:

- If a sentry node does not respond to a sentry request in a timely manner, the gateway node will move on to the next node in the list of sentry nodes
- If a node does not respond to the request for data from the gateway node at the end of *RemoteSensing*, it will be skipped and the gateway node will move on to the next node in line.

Step (4): Stopping *AutoMonitor*

AutoMonitor and *ThresholdSentry* can be stopped at any time to allow *RemoteSensing* or other commands and operations to be manually executed. This is done with the "`StopAutoMonitor`" command in the BluSH interface. The process can then be restarted from Step (1).

Troubleshooting Tips for *AutoMonitor and ThresholdSentry*

1. **Problem:** One or more of the sentry nodes is consistently unresponsive to sentry requests, i.e. the following debug output is displayed often for node X:

   ```
   Threshold not exceeded on node X
   ```

   Solution: Node X may be low on battery, be out of range or have other hardware problems. *AutoMonitor* should be stopped (`StopAutoMonitor`) and restarted with the problem node(s) omitted from the input file until they can be replaced/repaired.

2. **Problem:** One or more of the nodes consistently fails to report data during *RemoteSensing*, i.e. the following debug output is displayed often for node X:

   ```
   Data from node X unavailable; skipping and moving on.
   ```

   Solution: Node X may be low on battery, be out of range or have other hardware problems. *AutoMonitor* should be stopped (`StopAutoMonitor`) and restarted with the problem node(s) omitted from the input file until they can be replaced/repaired. Alternatively, the RC_DWAIT and RC_DWAITCONST flash constants can be increased to give the reliable communication protocol more time before timing out.

3.5 Utilities

In addition to the applications presented, there are a number of utility commands that may be executed at the BluSH command prompt from any ISHMP application. The utility commands that involve sending a command/request to a leaf node are implemented using the *RemoteCommand* utility while utility commands that involve the gateway node only are implemented using the *LocalCommand* utility. Some commands are implemented independently of the *RemoteCommand/LocalCommand* utilities. The following list describes the commands, their arguments and functionality they provide:

RemoteCommand/LocalCommand

These utilites are run by typing either *RemoteCommand* or *LocalCommand* followed by one of these command options: (example: `RemoteCommand Vbat 61`)

- Vbat [nodeId]
  Returns the reading from the VBAT pin on the gateway node or the requested leaf node. Note that VBAT = Actual battery voltage – 0.4V.
- ReadTemp [nodeId]
  Returns the temperature reading from the gateway node or the requested leaf node (°C).
- ReadLight [nodeId]
  Returns the light reading from the local or requested leaf node (lux).
- ReadHumidity [nodeId]
  Returns the humidity reading from the local or requested leaf node (percent).
- SetRadioChannel [nodeId] <channel>
  Sets the radio channel of the gateway node or the requested leaf node to the specified channel. Valid radio channels are 11 – 26.
- SetRadioPower [nodeId] <power>
  Sets the radio power of the local or requested leaf node to the specified power. If no nodeId is listed, it sets the gateway node's power. Valid power levels are 1-31.
- Reset [nodeId]
  Resets the gateway node or the specified leaf node.
- Sleep <sleep time> [nodeId]
  Puts the gateway node or the specified leaf node into deep sleep mode for sleep time seconds.
- ReadFC <index> [nodeId]
  Read Flash constant values for set *index* on the gateway node or the specified leaf node.
- RestoreFC <index> [nodeId] [nodeId] …
  Restore default Flash constant values for set *index* on the gateway node or the specified leaf nodes.
- WriteFC <index> [nodeId] [nodeId] …
  Write custom Flash constant values for set *index* on the gateway node or the specified leaf nodes.
- ChargeStatus [nodeId]
  Return the values of the Vbat, charging voltage(v-chg), charging current(i_chg) and charging status(chg–ctr), only when rechargeable battery and charging module(like solar panel) are used.
- SetSHMS <R2p> <R3p> <Gain> <V-offset> [nodeId] …
  SHM-S board parameter setting for strain measurement
- SHMSAutoBalance <sensing time in sec> <iteration #> [nodeId] …

Make a wheatstone bridge balance before sensing start

The following utility commands are available independent of the *RemoteCommand/LocalCommand* utilities are implemented on the gateway node only:

- SetUARTSpeed
  Sets the baud rate of the data UART port. Valid baud rates are 115200, 230400, 460800 and 921600.
- SetCoreVoltage
  Sets the gateway node's core voltage. Valid values are 850 – 1625 (mV).
- NodeID
  Returns the nodeId of the gateway node.
- GetFreq
  Returns the processor frequency of the gateway node.
- SwitchFreq <freq>
  Sets the processor frequency of the gateway node to the specified frequency. Valid frequencies are 13, 104, 208 and 416 (MHz).
- ResetWatchdog
  Resets the watchdog timer on the gateway node.
- GoToSleep
  Makes the current node sleep for specified amount of time
- ToolkitVersion
  Displays the version of the ISHMP Serive Toolsuite
- NodeID
  Displays the node Id of the gateway Imote2 node, which is locally connected
- ListNodes
  Displays the node IDs of the leaf nodes that are responsive.
- ResetWatchdog
  Resets the Watchdog timer that reset Imote2 after a predefined idle time
- TestRadio
  Assesses wireless communication performance by measuring the packet reception rate between two sensors
- RemoteTestRadio
  Assesses wireless communication performance by measuring the packet reception rate between designated remote Node and leaf node sensors.
- GetResetCause
  Displays what has caused the previous reset
- SetSHMS
  SHM-S board parameter setting for strain measurement
- SHMSAutoBalance
  automatic bridge balance of SHM-S board before sensing start

See Section 4.3 for more information on the `TestRadio` utility.

4 Deployment

The need for environmental hardening and the longer communication distances required by full-scale SHM deployments are just two of the considerations that should to be addressed when moving from the laboratory to the real world. The following sections provide guidelines on preparing for a full-scale deployment.

4.1 Battery voltage

The battery voltage for each Imote2 should be checked before testing. The nominal voltage of one AAA battery is 1.5 V, thus the nominal voltage supplied by three batteries is 4.5 V. Typically new AAA batteries initially have a voltage higher than the nominal voltage output. As a result, three new AAA batteries often exceed the 4.7 V maximum battery voltage allowed by the IBB2400 battery boards. If the voltage is too high, a safeguard mechanism on the battery board prevents it from supplying power to the Imote2. In such a case, using one slightly used battery with two newer batteries will reduce the voltage to within the allowable range. Other important tips are:

- Keep the voltage level between 3.7 and 4.7 V
    - If the voltage exceeds 4.7 V, the Imote2 will not turn on as discussed above.
    - If the voltage is less than 3.7 V, the ISM400 sensor board will not function.
    - If the voltage is less than 3.4 V, the Imote2 will not turn on.
- You can check the battery voltage of any Imote2 remotely using the `BluSh` utility command `RemoteCommand Vbat <nodeID>`. Note that the `Vbat` reading is between 0.3 V and 0.4 V less than the actual battery voltage, because this voltage is read after it passes the protection diode. Therefore, for operating nodes the `Vbat` reading should fall between 3.3 and 4.4 V.

4.2 External Antennas

By default, MEMSIC configures the Imote2 to use the onboard antenna. However, using an external antenna (e.g., the Antenova Mica 2.4 GHz SMD) is strongly recommended for full-scale deployments as they not only provide longer communication range but they also provide more consistent communication performance.[1] When using an external antenna, the following considerations should be taken into account for better communication:

- The antenna should be firmly screwed onto the connector to avoid unnecessary voltage losses.
- The radiation pattern of the external antenna resembles a torus, which is typical of a dipole antenna. Thus, if the antennas are parallel, the radiation patterns will overlap for improved communication performance (see

[1] Linderman, L.E., Rice, J.A., Barot, S., Spencer Jr., B.F., and Bernhard, J.T. (2009). "Characterization of Wireless Smart Sensor Performance." *ASCE J. of Eng. Mech*, to appear.

- Figure 25). Orienting the antennas perpendicular to one another should be avoided, as it results in poorer communication performance.
- Avoid encasing the Imote2 in steel or glass
- Elevate the sensors at least 2 feet from the ground when possible
- Keep line-of-sight between the gateway node and the leaf nodes when possible

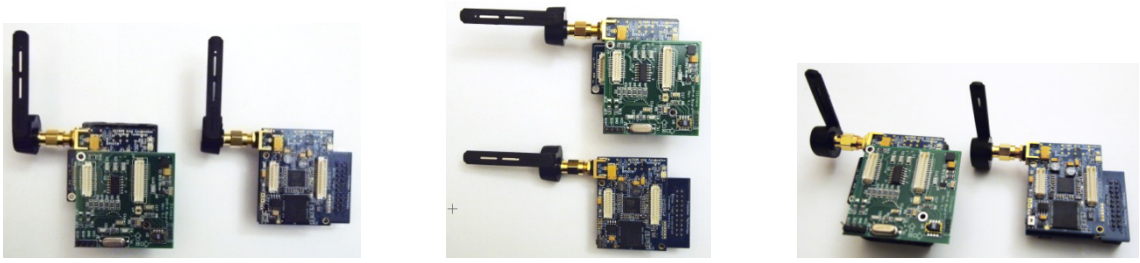


Figure 25. Recommended antenna orientations.

Note that orienting the antennas perpendicular to any metal surface (e.g., the flange of a steel girder) also can help to reduce the signal attenuation due to the presence of metal.

4.3 Assessing the Communication Environment with `TestRadio`

Each deployment site has a different communication environment that can have multiple sources of interference. For any successful deployment, an initial assessment of the communication range and Imote2 performance should be conducted. The `BluSH` utility command `TestRadio` is used to make this assessment by measuring the packet reception rate between two sensors. The results of `TestRadio` are useful in determining the optimal sensor placement. Leaf nodes that have better performance can be located farther from the gateway node, while leaf nodes whose performance is less than perfect need to be located close to a gateway node. This section provides an overview of how to use `TestRadio` to determine node-to-node communication performance.

4.3.1 Configuration test

In this step, you will place the nodes in the positions they will occupy in the final deployment.

- Position the gateway node in a fixed location that is as close as possible to the desired final location of the base station.
- Place the leaf nodes in their desired locations and turn them on.
- Position the antennas in what you believe to be their optimal configuration.

4.3.2 Run `TestRadio`

Because `TestRadio` does not collect data, it can be run in either the debug UART using `autocomm` with the interface board. Bring up a `BluSH>` prompt for the gateway node and run:

- `TestRadio <count> <nodeID> [nodeID] [nodeID]`

where `count` is the number of packets to be sent, and the `node IDs` are the IDs of up to ten of the leaf nodes being tested (between 100 and 1000 packets make for a good test). When the command is executed, the gateway node broadcasts the packets to the leaf nodes. The leaf nodes then transmit, in turn, the packets they successfully received back to the gateway node. The output of `TestRadio` is the round-trip packet reception rate. In `TestRadio`, the commands are sent reliably; the test will not fail due to the `TestRadio` command not being received unless communication between the gateway and leaf nodes in question is not possible. The test packets are not sent reliably thus allowing for raw transmission loss to be determined. Figure 26 shows the debug commands and output for a `TestRadio` run with two leaf nodes.

```
BluSH>TestRadio 1000 3
BluSH>Sending 1000 packets to 1 node(s): 3
Request successfully sent to 1 node(s).
Sending data messages...
Data messages sent.
Querying node 3...
Finished receiving responses from node 3.
addr    cnt     %       rssi    lqi     rcnt    r%      rrssi   rlqi
3       1000    100     18      107     1000    100     20      107
```

Figure 26. Sample command and output for `TestRadio`.

The results printed to the screen are:

- `addr` – the node ID of the node tested
- `cnt` – the total number of packets successfully returned from the leaf node back to the gateway node
- `%` – the percentage of packets that were successfully returned from the leaf node back to the gateway node
- `rssi` – received signal strength indicator (higher is better)
- `lqi` – link quality indicator (higher is better)
- `rcnt` – the number of packets that were successfully received by the leaf node
- `r%` – the percentage of packets successfully received by the leaf node
- `rrssi` – received signal strength indicator at the leaf node
- `rlqi` – link quality indicator at the leaf node

The round trip packet reception rate (`%`) gives an overall indication of the quality of the communication between each remote and the gateway node. Values above 90% will

perform well in most cases using the reliable communication protocol imbedded in the ISHMP software. The reliable communication protocol ensures that no data is lost, even when the packet reception rate is below 100%. If the packet reception rates determined by `TestRadio` are too low, the network topology should be reconfigured or the antenna positions changed to improve the reception rates.

4.3.3 Maximum communication range

Under ideal conditions, the Imote2 nodes using external antennas can communicate at distances of over 300m. Typical ranges when line-of-sight is possible are ~150m using point to point communication (i.e., without multi-hop communication). The communication range is directly related to the quality of the environmental conditions and sensor placement.

4.4 Radio Channel Selection

Selecting the proper communication channel for the network is critical in areas where an 802.11 (Wi-Fi) network is present. As illustrated in Figure 27, both Wi-Fi networks and the Imote2 operate in the 2.4 GHz band. Wi-Fi generally uses 802.11 channels 1, 6 and 11 because they do not overlap each other. Therefore, operating the Imote2s on channels 15, 20, 25 and 26 will cause the least interference with Wi-Fi. In the United States, Wi-Fi does not operate above channel 11 though in Japan channels 12 through 14 are also used. Channel 26 has been reported by ISHMP users to not be as reliable as other channels.

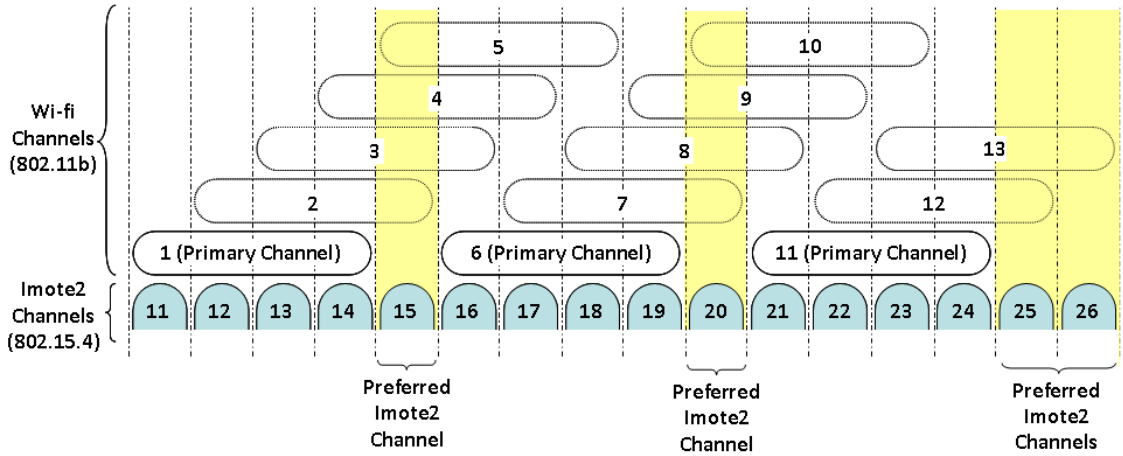


Figure 27. Wi-Fi and Imote2 channel overlap in the 2.4GHz frequency band.

Free networking software, such as NetStumbler (http://www.netstumbler.com/downloads/), can help you determine the Wi-Fi channels operating near your network. Using Figure 27, you can select the appropriate 802.15.4 channel for your Imote2 network to avoid interference with the observed Wi-Fi channels. In general, channel 25 on the Imote2 is the preferred operating channel and is set as the default in the ISHMP Services Toolsuite. The network's channel may be

changed any time using the `SetRadioChannel` command. In the command window, run:

- `autocomm -d COMy`

where COMy is the higher COM port. At the `BluSH>` prompt, run the following commands:

- `RemoteCommand SetRadioChannel <channel> <leaf nodeID>`

  This command sets the radio channel of the requested leaf node to the specified channel. The set of valid radio channels is 11 through 26 inclusive.

- `LocalCommand SetRadioChannel <channel>`

  This command sets the radio channel of the gateway node to the specified channel. Again, the set of valid radio channels is 11 through 26 inclusive.

Always set the channel for the leaf nodes first so when the local channel is changed the network can still communicate properly. If you set the gateway node's channel before the leaf nodes, you will not be able to communicate with the leaf nodes until you set the gateway node's channel back to its original setting. The Imote2 returns to the default radio channel when it is turned off/on or is reset.
The default radio channel may be changed in the Makefile prior to compilation and installation (see Figure 28). Make sure the channel is changed to the same channel in the Makefile for all nodes in the network otherwise they will not be able to communicate.

```
COMPONENT = RemoteSensingC

# supported sensorboards: ITS400CA, ITS400CB, SHM_A, SHM_H, SHM_DAQ
SENSORBOARD = SHM_A

# configuration parameters, uncomment only if changing default
values
#UART_SPEED = UART_BAUD_921600
#SHMA_CUSTOM_FILTER_RATE = 1000
#RFPOWER = 31
RFCHANNEL = 26      # Default channel '25' has changed to '26'

# optional components, uncomment to enable
USE_WATCHDOG = 1
#USE_CHARGER_CONTROL = 1
#USE_SNOOZE_ALARM = 1
#USE_MULTIHOP = 1

# required components
USE_SENSING_UNIT = 1

include $(SHMLIB)/ISHM/Makerules
```

Figure 28. Change of the default radio channel in the Makefile

Multiple Imote2 sub-networks can operate in the same location if different radio channels are selected for each sub-network.

4.5 Sensor Installation

Installing the Imote2s outdoors means that they must be protected from the elements through environmental hardening.

4.5.1 Environmental Enclosure

Prior to deploying a network of Imote2s outdoors, they must be housed in enclosures that protect against the elements like rain, wind, and dust. The enclosure should therefore have a lid with a rubber gasket to make it waterproof. The enclosure should house the batteries used in your deployment and an external antenna. Figure 29 gives an example of an enclosure that adequately houses the imote2, sensor board, battery board, and 3 D-cell batteries.

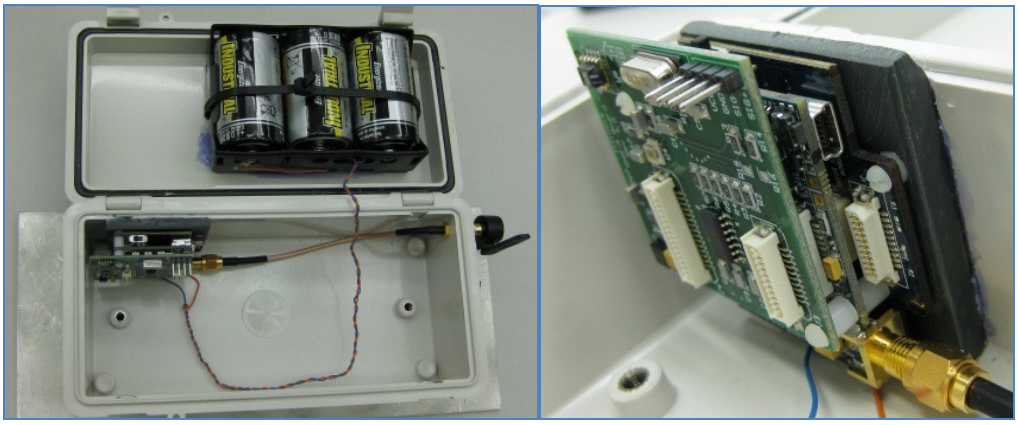


Figure 29. Sample environmental enclosure.

4.5.2 Antenna extension cable

When choosing an enclosure, an antenna extension cable (e.g., Emerson Network Power part # 415-0031-006) may be required to connect the Imote2 to the external antenna mounted on the enclosure wall. When needed, the extension cable should be as short and straight as possible as the cable can introduce significant signal attenuation.

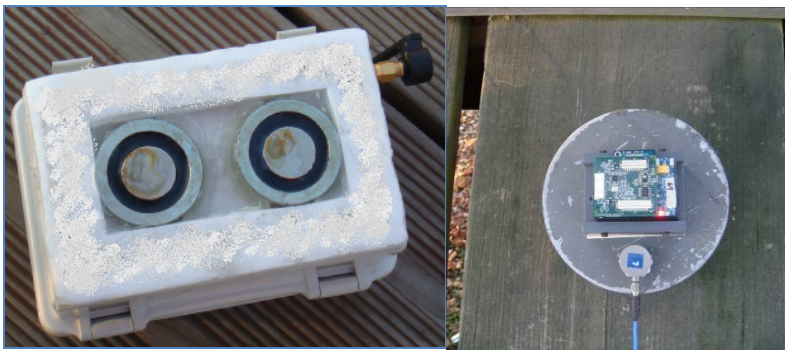
4.5.3 Mounting

The leaf nodes need to be properly mounted on the structure to get meaningful acceleration data. Possible mounting methods include:

- Clamping a mounting plate attached to the enclosure to the structure (see Figure 30.a).
- Using U-bends threaded through a mounting plate to attach the Imote2 enclosure to a cable (see Figure 30.b).

- Using magnetic mounts to attach the enclosure to steel structures (see Figure 30.c.). Tests have shown that the magnet has little effect on the Imote2's communication performance.
- For temporary attachment, using hot glue to attach an Imote2 in or out of its enclosure has proven effective and useful (see Figure 30.d).

a) using clamps b) using U-bend (for cable)



c) using magnets c) hot gluing to weight

Figure 30. Sample examples of mounting methods.

5 Troubleshooting

The following list gives some guidelines that will help improve the success of your Imote2 testing and deployment.

- If a node is exhibiting unexpected behavior, check the battery voltage. It is easy to forget and can lead to frustration if overlooked.
- If a node is repeatedly exhibiting unexpected behavior and the battery voltage is adequate, it could simply be a hardware problem. Imote2s, battery boards, sensor boards and antennas have finite lives and can simply be "bad". Try to isolate the source of the hardware problem by switching the battery board/sensor board/antenna.
- If after uploading the application image, the mote continuously reboots (the LED flashes briefly about once every 2 seconds). This occurs when incorrect Flash constants are loaded on the mote. It is necessary to reprogram the mote with a fresh set of Flash constants. See the README file in tools/WriteFlashConstants for detailed instructions.
- If you experience consistent difficulty with communication (network wakeup times out, sending *RemoteSensing* channel parameters takes too long, etc) check the communication environment using *TestRadio*. This will reveal if a particular node has communication problems or if the test environment/network topology in general is not conducive to successful communication.
- If you have problems with printing data files, such as missing or corrupt output, you may need to reduce the port speed used by the application. This consists of two steps: in the Imote2's BluSH shell, run "`LocalCommand SetUARTSpeed 460800`" before starting the *RemoteSensing* application; on the PC, run "`autocomm –b 460800 –n –o out.txt COMx`".
- If printing problems persist after the above change, you may need to disable some hardware drivers or system tray applications to alleviate the problem. In particular, applications relating to USB devices (cameras, printers, music players) have been known to cause problems. In some rare cases even with the minimal number of processes running the PC may not be fast enough to print the data files correctly and a different PC may need to be used.
- If you plan to deploy your network in cold temperatures, be aware that battery performance degrades as the temperature decreases and this may cause problems with the operation of the Imote2s.

6 Conclusion

This User's Guide has provided instructions on how to acquire high-quality sensor data from a network of Imote2s utilizing the ISHMP services toolsuite. In addition, considerations for network deployment have also been outlined. While the troubleshooting tips are not comprehensive, they address many commonly encountered difficulties. With the experience gained from following the instructions and tips in this guide, you will have the ability to deploy a network of Imote2s for a wide range of monitoring applications.

Please check back often with the ISHMP website (http://shm.cs.uiuc.edu) for software and documentation updates.

Glossary

- Baud rate – the UART communication speed between the interface/debug board (IIB) and the PC
- Comment/uncomment – a line or section of code may be commented out (i.e. disabled) by inserting special characters at the beginning of the line. In the Makefiles, the special character is "#".
- BluSH – short for Blue Shell, this is the command line shell interface that is used to communicate with the Imote2.
- UART - Universal asynchronous receiver/transmitter. The UART is hardware that provides a serial port connection for communication with Imote2 from the PC.
- Channel mask – an input parameter representing the channels to be sampled as a single value. For example, to request channels 1 and 3 the channel mask is 13. To request channels 1, 2 and 3 the channel mask is 123, etc.
- ISHMP – Illinois Structural Health Monitoring Project (http://shm.cs.uiuc.edu).
- Compile – to translate the higher level code (C and NesC) into an executable image (machine language) for the Imote2 processor
- Log files – files that contain a log of the debug commands and output generated from the debug UART; not data files.

For inquiries, please contact:

Professor B.F. Spencer, Jr.
bfs@illinois.edu
University of Illinois at Urbana-Champaign
Department of Civil and Environmental Engineering
2213 Newmark Civil Engineering Laboratory, MC-250
205 North Mathews Ave
Urbana, IL 61801
USA

Or visit:

http://vibration.shef.ac.uk/imote2_forum
http://shm.cs.uiuc.edu