Getting up and Running with TinyOS

This guide walks you through the installation, verification, compilation and running the TinyOS application on Windows based PC. There are several key steps to getting up and running with TinyOS.

Get a copy of the TinyOS development environment

- Insert the TinyOS CD into the CDROM drive. It is self-installing application and installs the following utilities on your hard drive.
  - **cygwin** development tools that provide Unix-like functionality under Windows. This leaves you with a cygwin icon on the desktop.
  - **avrgcc** (avr utility) that will install the compiler and bin utilities.
  - **JDK** (java utility) to read the serial port.
  - A copy of **TOS** distribution utilities in a folder called **nest**.
- If you do not have the installation CD, go to http://today.cs.berkeley.edu/tos/download.html and the click on the appropriate link for platform-specific install instructions.

For the latest versions of the above utilities, please visit http://today.cs.berkeley.edu/tos

Confirming your setup

When working with embedded devices, it is very difficult to debug applications. Because of this, you want to make sure that the tools you are using are working properly and that the hardware is functioning correctly. This will save you countless hours of searching for bugs in your application when the real problem is in the tools. This section will show you how to check your system and the hardware.

Confirm that the TinyOS tools are installed correctly:

First, we will check that the tools have been installed correctly and that the environment variables are set. "*toscheck*" is a script that will perform this functions.

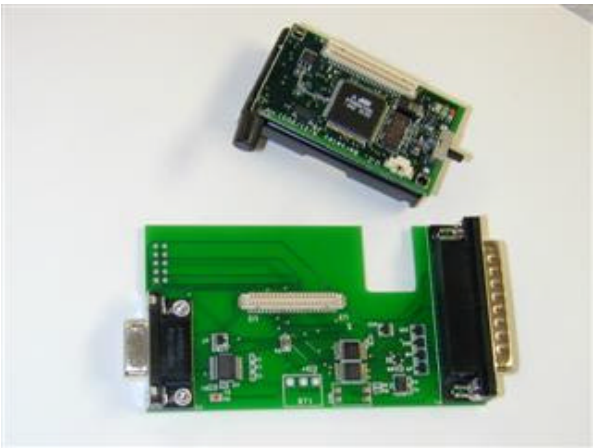
- Run the cygwin application by double-clicking the icon that can be found on your desktop.
- Change into your nest/tools directory and type "*toscheck*".
- The last line of the output should be "*toscheck completed without error*".

Confirm that the TinyOS hardware is working:

To test the hardware, we have provided an application: *mica_hardware_verify*. It is only for use on the mica platform.

- Change to the **/nest/apps/mica_hardware_verify** directory and type "*make mica*."
- The compilation process should complete without any errors. If it compiled correctly it will print out a profile of the memory used by the application.

- Next install the application onto a mica node. Place a powered-on node into a programming board. The red LED on the programming board should light.
- Connect the programming board to the parallel port of your computer. Type "*make mica install*". This will load the program onto the device. The output should end with "*Atmel AVR ATmega103 is found. Verifying: flash*". Now you have verified that the programming tools and the computer's parallel port are working correctly.



Mica mote next to the programming board

Mica mote connected to the programming board

- The next step is to verify the mote hardware. First, confirm that the LEDs are blinking like a counter.
- Next connect the programming board to the serial port of the computer. The hardware verify program will send data over the UART that contains it status. To read from the serial port compile and run "*hardware_check.java*" by using the following commands (Using COM1):
  - Type "*javac hardware_check.java*".
  - When done, at the command prompt type "*java hardware_check COM1*".
  - This program is checking the serial ID of the mote, the flash connectivity, the UART functionality and the external clock. All status reports (UART transmission and packet reception) should be successful or positive.
  - To stop the transmission use Ctrl + Break.

Confirm that the radio is working:

To do this, you will need two nodes.

- Take a second node (that has passed the hardware check up to this point) and install it with *generic_base_high_speed* by using the following steps:

  - Change to the **/nest/apps/generic_base_high_speed** directory and type "*make mica*".

- Connect the programming board to the parallel port of your computer. Type "*make mica install*". This will load the program onto the device.
- This node will act as a radio gateway to a second node.

- Once installed, leave this node in the programming board and place the original node next to it.
- Re-run the hardware_check java application as explained in above section.
- You should now see both the RF transmission and the RF reception is successful. Note that some of the packets received will not pass the crc check. These packets should be ignored.

Introduction to TinyOS

Following section introduces the basic TOS concepts through a very simple example that periodically blinks the LEDs. The application can be found at **nest/apps/blink**. This walks you through the major concepts required to program Tiny OS applications. These include a description of components, frames, commands, and events. The Tiny OS programming model is explained. The role of each of the different file types is detailed. The heart of this application is provided by a single component, blink.

Components:

A *TOS component* consists of an *implementation* and an *interface*, described by a `.c` file and a `.comp` file, respectively, here **nest/apps/blink/blink.c** and **nest/apps/blink/blink.comp**.

A *TOS component implementation* may contain a frame, a set of event handlers, a set of commands, a set of tasks, and local procedures. The commands must match the commands it claims to accept in the interface and similarly the events must match the events it claims to handle. The external namespace of the implementation is the commands it uses and events it signals. The implementation must include *tos.h* and *NAME.h*, where NAME is the name of the component.

A *TOS component interface* specifies the name of the component module, describes the set of commands the component accepts, the commands that it uses, the events that it handles, and the events it signals. The implementation is written in terms of the interface namespace. The syntax of the interface file is:

```
TOS_MODULE name;
ACCEPTS{
      command_signatures
};

HANDLES{
      event_signatures
};
```

```
USES{
     command_signatures
};

SIGNALS{
     event_signatures
};
```

Frames:

A *TOS frame* contains all static variables available to the implementation of the component. Variables within the frame are referenced using the VAR macro.

Looking at the implementation itself, **nest/apps/blink/blink.c** consists of a frame, two commands and one event handler. The frame contains a single static variable, state. The two commands support a simple initialization protocol. By convention, we provide a MAIN component, which starts up on hardware reset and invokes an INIT command followed by a START command. This allows the collection of components forming an application to come up cleanly.

Commands:

A *TOS command* is declared as char TOS_COMMAND(cmd_name)(cmd_param_list). The return value is the status of the request: 0 = failure. A command may call commands, or post tasks, but may not signal events. It must complete in a bounded, but not necessarily short, amount of time. It cannot block or spin waiting for state changes.

A TOS command is called using TOS_CALL_COMMAND(cmd_name)(cmd_args_list)

The BLINK_INIT command turns off all the LEDs and initializes its frame. It then initializes subcomponents, passing an argument that indicates it should receive a clock event at 1 Hz. The mapping from simple clock period to hardware specific values is provided by **nest/tos/platform/NAME/include/hardware.h**.

The BLINK_START command does nothing more in this case. After the initialization phase, this component is merely asleep until events occur.

Events:

A *TOS event* is declared as char TOS_EVENT(evnt_name)(evnt_arg_list). Lowest level events are connected directly to hardware interrupts by the system hardware abstraction layer. It does key work before re-enabling interrupts. An event may signal events, call commands, or post tasks. It must complete in a short amount of time, bounded by the jitter requirements of the overall application.

A TOS event is signalled using `TOS_SIGNAL_EVENT(evnt_name)(evnt_arg_list)`

The BLINK_CLOCK_EVENT is intended to handle clock events on a periodic basis. It utilizes an internal state variable to keep track of the state of the red LED and toggles the LED on each clock tick.

Looking at the interface definition, **nest/apps/blink/blink.comp**, we see that it state the module name BLINK, declares the two commands it accepts and the one event it handles. These declarations match the signatures of the associated procedures in the implementation. It also declares that it uses a SUB_INIT command and a family of LED commands. It signals no events.

TinyOS Applications

A *TOS application* consists of a graph of components. A textual representation of this graph is contained in the description file for the application, here **nest/apps/blink/blink.desc**.

The TinyOS component approach separates creation of the components from their composition. It makes it very easy to swap components in and out and interpose components. It also limits the interactions between components to very narrow channels.

A *TOS description file* specifies the set of component modules used in the application and the wiring of commands and events across component interfaces. The component module name should match the root of the interface and the implementation file, case sensitive, as well as the module name in the .comp file. The wiring must be a subset of the interface elements associated with the components. There is a search path associated with resolution of module, typically consisting of *".*, *tos/system*, *tos/platform*, *tos/shared*". The syntax is as follows.

```
include modules{
    module list
};

connection list
```

Compile your first program

- Run the cygwin application by double-clicking the icon that can be found on your desktop.
- Enter the **nest/apps/blink** directory using your shell (cygwin under Windows); it is a good application to make sure that the most basic hardware is working.
- Type "*make mica*" in the shell. This should complete successfully and create a binary image of your program for the mica motes.
- All objects, generated includes, and executables are place in the bin directory for the specific platform, eg, **`binmica/`**
- You should, of course, observe errors and warnings that arise in building your application. This example should not have any. At the very end, the Make shows you a piece of the load map that tells you whether your application fits.

Downloading Programs to the Device

- To download your program into the Mica wireless sensor node, place the Mica board (or board and sensor stack) into the bay on the programming board, as shown above.
- You can either supply a 3 V supply to the connector on the programming board or power the node directly. The red LED labeled D2 on the programming board will be on when power is supplied.
- Plug the 32-pin connector into the parallel port of a laptop configured with the TOS tools, or connect use a standard mail-female DB32 parallel port cable.
- Type "*make mica install*". If you are using windows and the install doesn't work, you make need to fiddle with the port specified to uisp; depending on the hardware, cygwin can map parallel ports to wildly different names (use the -dlpt= option).

You should see the upload take place (this will take 2 minutes or so) and the red LED should blink on and off and 0.5 Hz.

Note: make clean will clean up all generated files. If your directory seems cluttered, it is often a good idea to run make clean.

Reading from the COM port

The goal of this section is to close the loop on communicating with the motes and bring the data onto a PC. The first stage of this will be to display the data coming from the mote in a text console. We will then extend this to demonstrate the display of this data on a graphical interface. The program to be run on the motes is **nest/apps/oscilloscope.** The application consists of a single module that reads data from the ADC and sends out packets of the sensor reading of data channel 1 to the to the serial port. This application does not transmit the data over the radio. However, it would be simple to extend it to do so. It is intended to be used with the basic sensor board.

Displaying Data onto the screen:

- Enter the **nest/apps/oscilloscope** directory and type "*make mica*".
- To install the application onto a mica node, plug the programming board to the parallel port of your computer and type "*make install mica*".
- Once the application is installed, each time the yellow LED blinks, a packet will be send to the serial port. Additionally, the RED led will be turned on when the sensor reading is greater that 0x20. In normal lighting, you should see the red led on. If you place the mote in darkness (very dark) then you should see the red led go off.
- Switch to the **nest/tools** directory and run "*make*".
- Connect the serial port on the programming board to your machine and then type "*java listen COM1*" (assuming the mote is programmed into COM1).

The application that you are running is simply printing out the packets that are coming form the mote. Each data packet that comes out of the mote contains several fields of

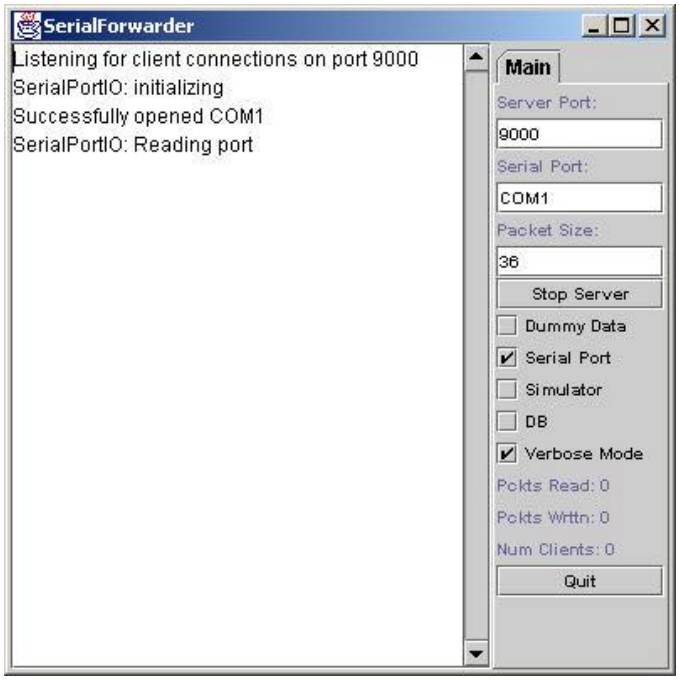
data. They include: the source ID, the reading number of the first reading, the ADC channel that the data is coming from and 10 sensor readings.

Starting the SerialForwarder:

Listen is the most basic way of communicating with the mote. It is a program that directly opens the serial port and prints the data that it read out to the screen. While you will be able to clearly see a changing pattern in the data (try covering and uncovering the sensor), it is difficult to visualize the sensor readings.

Along with the mote code, we also provide a PC application designed to visualize the data coming from a mote. However to use the GUI, you have to switch over to using our SerialForwarder, a tool that abstracts away the serial port from an application.

- Go into the **nest/tools** directory and type *"javac net/tinyos/SerialForwarder/SerialForward.java"*. This file contains a Java program that presents a unified abstraction of reading mote output. In addition, you need to run a program that connects to the SerialForwarder and presents data to you in a meaningful form.
- To run the serial forwarder, remain in the tools directory and type "*java net.tinyos.SerialForwarder.SerialForward*". This will open up a GUI window and will also display text in the shell that started it. Check both locations to make sure that no errors occurred. Make sure that it states that the serial port was successfully opened. The window should look like.

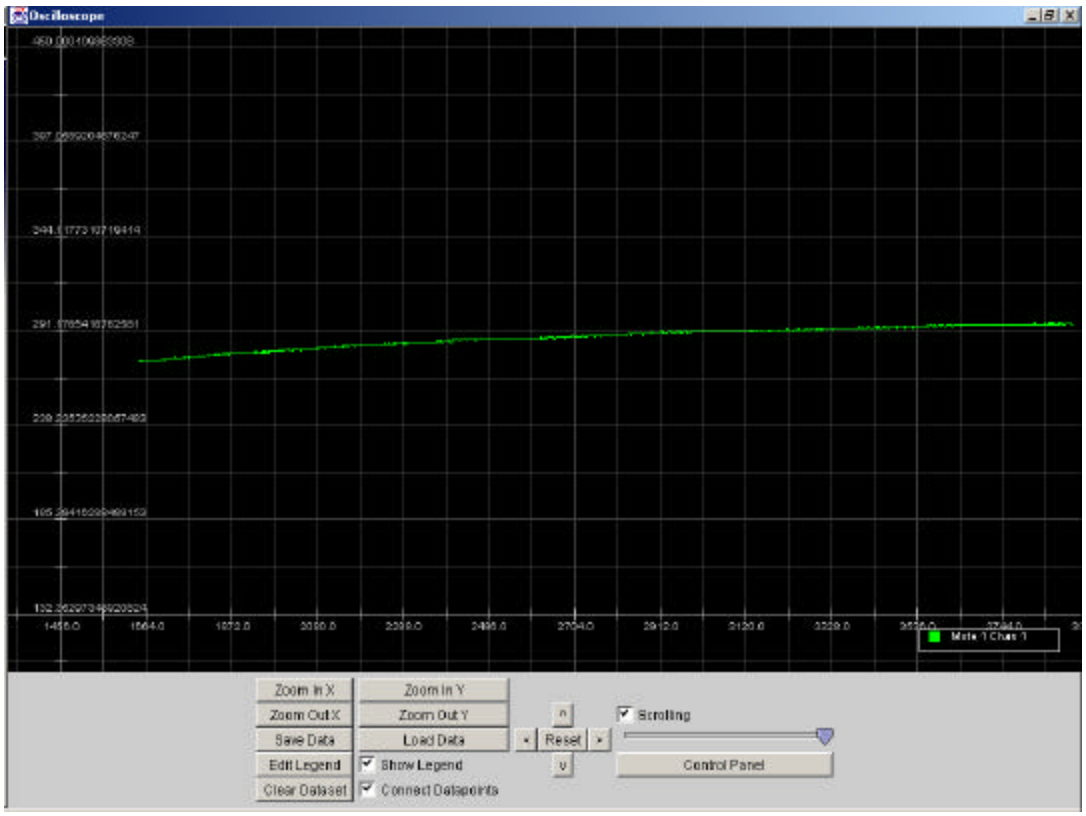


As packets arrive from the mote connected to the serial port, you will see the "Pckts Read:" field in the lower right corner begin to increment.

Starting the Oscilloscope GUI:

It is now time to graphically display the data coming from the motes.

- In a new window, return to the tools directory and type "*java net.tinyos.oscilloscope.oscilloscope*". A graphical window will popup and begin displaying the data coming from your mote.
- This application is actually connecting to the serial forwarder, not directly to you serial port. This allows you to simultaneously run other applications that are communicating with a mote.



Installing and Executing Surge Demo

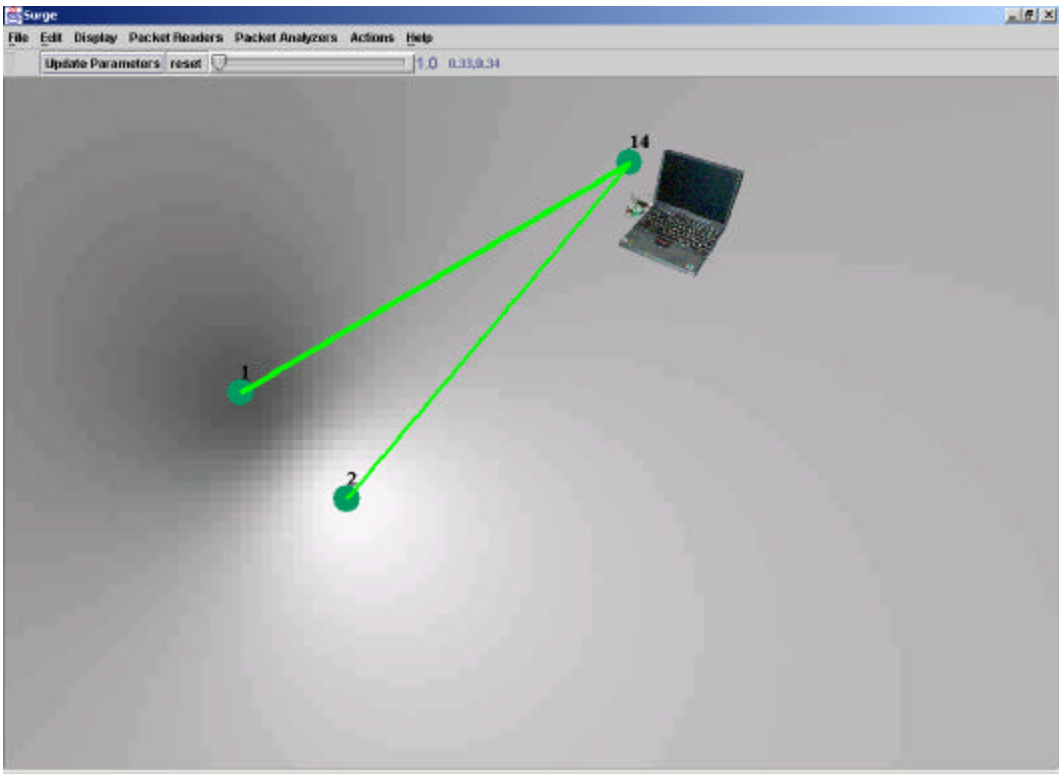
Surge Demo is a highly multi-threaded and event-driven ad-hoc networking visualization tool. This demo application is not provided in the TinyOS installation CD. You would need to obtain this special application from TinyOS support group. Once you acquire this

Save the attachment "*congress-demo-2-20-02.tgz*" on to your hard drive (e.g. **c:\curgedemo**).

- Start cygwin and go to **c:/surgedemo** and then type command "*tar –zxf congress-demo-2-20-02.tgz*". This unzips the contents in the zip file in Unix environment under the folder **c:/surgedemo**.
  Note: For unzipping this file, do not use Windows or Winzip. The file was zipped in Unix environment and hence has to be unzipped using Unix using *tar* command.
- From here onwards to load the code into the Mica Motes, you should use these new tos utilities that can be found under **surgedemo/nest.**
- Load the Surge demo code in transmitter Mica(s):
  - With programming board plugged into parallel port of the PC, connect one of the nodes to the programming board.
  - Go to **surgedemo/nest/apps/surge** directory.
  - Type command "*make install.x mica*", where *x* is the id# that you want to assign to nodes which has to be unique (e.g. 1, 2, 3 etc).
  - Repeat the above step for different Mica motes; make sure that there is no conflict of id#.
- Load the Generic Base in the Receiver Mica:
  - With Programming board plugged into parallel port of the PC, connect a new node to the programming board.
  - Go to **surgedemo/nest/apps/generic_base_high_speed** and type "*make install.14 mica*"(the generic base is always identified with id# 14).
  - This node will now act as a radio gateway to the other nodes.
- Once installed, leave this node in the programming board and spread around the rest of nodes with sensor boards connected (turn on the battery).
- Executing the Surge Demo Application
  - Go to **surgedemo/surge2.0** directory and type command "*make*"
- Once this happens you should see the LEDs on the Mica to be blinking and the different Motes with specific id# should appear on the screen.

If you cover the light sensor with the hand, that particular Mote should get dark on the screen. You can make a particular Mote disappear from the screen by powering it off or move it farther away. The solid green lines indicate the active data transmission link and the red line represents previously active communication links, which has now become inactive, because the node found a better transmission path.

Note: The location of the node id#s on the screen does not represent the physical location of the nodes in the neighborhood of the PC. Be advised that the graphical interface for the Surge Demo is not very robust.



Learning more about TinyOS

Now that you have TinyOS working, you can start learning about it and writing your own programs. In the **nest/doc/tutorial** directory are the first lessons for learning TinyOS. Read and follow the instructions. There are also some documents in **nest/doc**, such as *tos-developer.pdf*, that provide some high level information on the OS structure.

SUNSTAR 商斯达实业集团是集研发、生产、工程、销售、代理经销 、技术咨询、信息服务等为一体的高科技企业，是专业高科技电子产品生产厂家，是具有 10 多年历史的专业电子元器件供应商，是中国最早和最大的仓储式连锁规模经营大型综合电子零部件代理分销商之一,是一家专业代理和分銷世界各大品牌 IC 芯片和電子元器件的连锁经营綜合性国际公司,专业经营进口、国产名厂名牌电子元件，型号、种类齐全。在香港、北京、深圳、上海、西安、成都等全国主要电子市场设有直属分公司和产品展示展销窗口门市部专卖店及代理分销商,已在全国范围内建成强大统一的供货和代理分销网络。 我们专业代理经销、开发生产电子元器件、集成电路、传感器、微波光电元器件、工控机/DOC/DOM 电子盘、专用电路、单片机开发、MCU/DSP/ARM/FPGA 软件硬件、二极管、三极管、模块等，是您可靠的一站式现货配套供应商、方案提供商、部件功能模块开发配套商。商斯达实业公司拥有庞大的资料库，有数位毕业于著名高校——有中国电子工业摇篮之称的西安电子科技大学（西军电）并长期从事国防尖端科技研究的高级工程师为您精挑细选、量身订做各种高科技电子元器件，并解决各种技术问题。

更多产品请看本公司产品专用销售网站:

商斯达中国传感器科技信息网：http://www.sensor-ic.com/

商斯达工控安防网：http://www.pc-ps.net/

商斯达电子元器件网：http://www.sunstare.com/

商斯达微波光电产品网:HTTP://www.rfoe.net/

商斯达消费电子产品网://www.icasic.com/

商斯达实业科技产品网://www.sunstars.cn/

传感器销售热线:

地址：深圳市福田区福华路福庆街鸿图大厦 1602 室

电话：0755-83370250 83376489 83376549 83607652 83370251 82500323

传真：0755-83376182 （0）13902971329 MSN: SUNS8888@hotmail.com

邮编：518033 E-mail:szss20@163.com QQ: 195847376

深圳赛格展销部：深圳华强北路赛格电子市场 2583 号 电话：0755-83665529 25059422

技术支持: 0755-83394033 13501568376

欢迎索取免费详细资料、设计指南和光盘 ；产品凡多，未能尽录，欢迎来电查询。

北京分公司：北京海淀区知春路 132 号中发电子大厦 3097 号

TEL：010-81159046 82615020 13501189838 FAX：010-62543996

上海分公司：上海市北京东路 668 号上海赛格电子市场 2B35 号

TEL：021-28311762 56703037 13701955389 FAX：021-56703037

西安分公司：西安高新开发区 20 所(中国电子科技集团导航技术研究所)

西安劳动南路 88 号电子商城二楼 D23 号

TEL：029-81022619 13072977981 FAX:029-88789382