TinyOS community

- [http://www.tinyos.net/](http://www.tinyos.net/)

TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. A worldwide community from academia and industry use, develop, and support the operating system as well as its associated tools, averaging 35,000 downloads a year.

Latest News

**January, 2013:** The transition to hosting at GitHub is now complete. Part of this transition includes slowly retiring TinyOS development mailing lists for bug tracking and issues to using the GitHub trackers. Thanks to all of the developers who are now improving TinyOS and requesting pulls!

**August 20, 2012:** TinyOS 2.1.2 is now officially released; you can download it from the debian packages on tinyos.stanford.edu. Manual installation with RPMs with the instructions on docs.tinyos.net will be forthcoming. TinyOS 2.1.2 includes:

- Support for updated msp430-gcc (4.6.3) and avr-gcc (4.1.2).
- A complete lowpan/RPL IPv6 stack.
- Support for the ucmini platform and ATmega128RFA1 chip.
- Numerous bug fixes and improvements.

FAQ
- Frequently asked questions about TinyOS

Learn
- Download TinyOS and learn how to use it

Community
- TinyOS Working Groups, mailing lists, and TEPs
Telosb / Tmote Sky

- CC2420 radio compatible with IEEE 802.15.4
- 250kbps data rate
- TI MSP430 microcontroller
  - 8MHz, 10kB RAM, 48k Flash
- Integrated antenna
  - Range 50m (indoor), 125m (outdoor)
- Integrated light, temperature, IR, humidity sensor
NesC

- Network Embedded System C
- Variation of C language
- Static language
  - No function pointers and no dynamic memory allocation

:nesc:nc

App.c
nesC Compiler
C Compiler
Main.exe

TinyOS Programming, Philip Levis
TinyOS Installation

- TinyOS 2.1.2 Installation
  - Linux, Window, OSX

- Required Software
  - msp-430 tools
    - msp430-libc, binutils-msp430, gcc-msp430
  - NesC: [https://github.com/tinyos/nesc.git](https://github.com/tinyos/nesc.git)
  - TinyOS: [https://github.com/tinyos/tinyos-main.git](https://github.com/tinyos/tinyos-main.git)
Connect motes

- **Check your TinyOS installation**
  - `tos-check-env`

- **Check which port a mote attached to**
  - `motelist`

```
dolvaragunatilaka@ubuntu:/opt/tinyos-2.1.2/apps$ motelist
Reference   Device            Description
----------   -----------------   -----------------
M4A6J3UG    /dev/ttyUSB0      Moteiv tmote sky
```

- **Give access permission**
  - `sudo chmod 666 /dev/<devicename>`
  - `sudo gpasswd -a username dialout`
TinyOS includes Makefiles to support the build process

```make
COMPONENT=MainAppC
TINYOS_ROOT_DIR?=../..include
$(TINYOS_ROOT_DIR)/Makefile.include
```

Compile an app without installing it on a mote:
- `make [platform]`
- Platform: telosb, micaz, mica2

Install an app on a mote:
- `make [re]install.[node ID] [platform]`

Remove compiled files:
- Make clean
Build Stages

**make install.1 telosb**

```bash
make install.1
```

```
nc -p build/telosb
nc -p build/telosb/main.exe
c -fsnc -separator= -Wall -Wshadow -Wnesc-all -target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM
GROUP=0x22 -DIDENT_APPNAME="BlinkAppC" -DIDENT_USERNAME="dolvaraguntillaka" -DIDENT_HOSTNAME="ubuntu" -DIDENT_USERHASH=0x114b2df8L -DIDENT_TIMESTA
MP=0x54c3fcb1L -DIDENT_UIDHASH=0xa335d9e1L BlinkAppC.nc -lm
compiled BlinkAppC to build/telosb/main.exe
2538 bytes in ROM
56 bytes in RAM
```

```
make install.1 telosb
```

```
dolvaraguntillaka@ubuntu:/opt/tinyos-2.1.2/apps/Bl
```

```
nc -p build/telosb
nc -p build/telosb/main.exe
c -fsnc -separator= -Wall -Wshadow -Wnesc-all -target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM
GROUP=0x22 -DIDENT_APPNAME="BlinkAppC" -DIDENT_USERNAME="dolvaraguntillaka" -DIDENT_HOSTNAME="ubuntu" -DIDENT_USERHASH=0x114b2df8L -DIDENT_TIMESTA
MP=0x54c3fcb1L -DIDENT_UIDHASH=0xa335d9e1L BlinkAppC.nc -lm
compiled BlinkAppC to build/telosb/main.exe
2538 bytes in ROM
56 bytes in RAM
```

```
writings TOS image
tos-set-symbols --objccopy msp430-objectcopy --objdump msp430-objdump --target ihex build/telosb/main.hex out-1 TOS_NODE_ID=1 Activ
essageAddressC__addr=1
```

```
Found mote on /dev/ttyUSB0 (using bsl,auto)
installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB0 -r -e -i -p build/telosb/main.hex.out-1
```

```
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Mass Erase...
```

```
MSP430 Bootstrap Loader Version: 1.39-goodfet-8
Mass Erase...
```

```
Transmit default password ...
Invoking BSL...
Transmit default password ...
```

```
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
```

```
Program ...
2598 bytes programmed.
Reset device ...
```

```
rn -f build/telosb/main.exe.out-1 build/telosb/main.hex.out-1
```

- nc to .c and .c to binary
- Set node ID
- program mote
Sensor Network Architecture

- Sensors
- Base station
- Gateway
- Internet
TinyOS Design

- Component-based architecture
  - Components and interfaces

- Task and event-based concurrency
  - Task: deferred computation
  - Events: preempt the execution of a task or another event.

- Split-phase operation
  - Command returns immediately
  - Event signals completion
TinyOS Execution model

- To save energy, node stays asleep most of the time
- Task and event based concurrency:
  - Computation is kicked off by hardware interrupts
  - Interrupts may schedule tasks to be executed at some time in the future
  - TinyOS scheduler continues running until all tasks are cleared, then sends mote back to sleep
Components

- NesC application consists of one or more components
- A component *provides* and *uses* interfaces
- Components defined two scopes:
  - Modules: implementation of interfaces
  - Configurations: wiring interfaces of a component to interfaces provided by other components

```plaintext
configuration BlinkAppC
{
    provide interfaces
}
Implementation
{
    ...
}

module BlinkC
{
    provides interfaces
    uses interfaces
}
Implementation
{
    ...
}
```
Interfaces

- List of one or more functions

1. Generic interface
   - Take one or more types as a parameter

```c
interface Queue<t> {
    ...
    command t head();
    command t dequeue();
    command error_t enqueue(t newVal);
    command t element(uint8_t idx);
}
```

```c
module QueueUserC {
    uses interface Queue<uint32-t>;
}
```
2. Bidirectional

- Commands and Events
- Users call commands and providers signal events.

```c
interface Receive {
    event message_t * Receive(message_t * msg, void * payload, uint8_t len);
    command void * getPayload(message_t * msg, uint8_t * len);
    command uint8_t payloadLength(message_t * msg);
}
```
Modules provide the implementation (logic) of one or more interfaces

They may use other interfaces:

```plaintext
module ExampleModuleC
{
    provides interface SplitControl;
    uses interface Receive;
    uses interface Receive as OtherReceive;
}
implementation
{
    ...
}
```

**Rename** interfaces with the *as* keyword -- required if you are using/providing more than one of the same interface!

- Variable declarations
- Helper functions
- Tasks
- Event handlers
- Command implementations
Placed inside `implementation` block like standard C declarations:

```c
... implementation {
    uint8_t localVariable;
    void increment(uint8_t amount); // declaration

    ...

    void increment(uint8_t amount) { // implementation
        localVariable += amount;
    }
}
...```
Modules: Tasks

- Look like functions, except:
  - Prefixed with `task`
  - Cannot return anything or accept any parameters

- Tasks are scheduled using the `post` keyword

- Can be preempted by interrupts, but not by other tasks
  - Design consideration: Break a series of long operations into multiple tasks

```cpp
implementation {
    ...
    task void handlePacket() {
    }
}
```

```cpp
post handlePacket();
```

Can post from within commands, events, and other tasks
Commands and events also look like C functions, except:

- they start with the keyword `command` or `event`.
- the “function” name is in the form: `InterfaceName.commandOrEventName`

```c
implementation {
    command error_t SplitControl.start()
    {
        // Implements SplitControl’s start() command
    }

    event message_t * Receive.receive(message_t * msg, void * payload, uint8_t len)
    {
        // Handles Receive’s receive() event
    }
}
```
Modules: Commands and Events

- Commands are invoked using the `call` keyword:

  ```
  call Leds.led0Toggle();
  // Invoke the led0Toggle command on the Leds interface
  ```

- Event handlers are invoked using the `signal` keyword:

  ```
  signal SplitControl.startDone();
  // Invoke the startDone event handler on the SplitControl interface
  ```
Component Scope - Configurations

Connect components / wire interfaces

```plaintext
configuration NetworkHandlerC
{
    provides interface SplitControl;
    uses interface Receive;
}

implementation
{
    components NetworkHandlerC as NH, 
    ActiveMessageP as AM;
    NH.Receive -> AM.Receive;
    SplitControl = NH.SplitControl;
}

--> wire to external interface
= wire to internal interface
```
Concurrent Model

- Task
  - deferred execution, run to completion
  - Does not preempt each other

- Event handler
  - Signal asynchronously by HW interrupt
  - Preempt tasks and other event handlers
  - Command/event uses `async` keyword

- Race condition: concurrent interrupt/task updates to shared states
Race conditions

1. Keep code synchronous (update shared state using task)
   - If timing isn’t crucial, defer code to tasks (synchronous)

```c
#include <cstdint>

uint8_t sharedCounter;

void task incrementCounter() {
    sharedCounter++;
}

async event void Alarm.fired() {
    post incrementCounter();
}

event void Receive.receive(...) {
    ...
    sharedCounter++;
}
```

Task is scheduled immediately, but executed later
2. **Atomic Block**

- Interrupts are disabled – use sparingly and make it short

```c
implementation {
    uint8_t sharedCounter;

    async event void Alarm.fired() {
        atomic{
            sharedCounter++;
        }
    }
}
async event void Alarm2.fired() {
    atomic{
        atomic{
            sharedCounter++;
        }
    }
}
```
Race Condition

- Compiler detects race condition -> false positive
- Absolutely sure that there is no race condition (or do not care if there is), use the `norace` keyword:

```c
implementation {
    norace uint8_t sharedCounter;

    async event void Alarm1.fired() {
        sharedCounter++;
        call Alarm2.start(200);
    }

    async event void Alarm2.fired() {
        sharedCounter--;
        call Alarm1.start(200);
    }
}
```

Race condition is impossible; these Alarms are mutually exclusive
Network Communication

- Each node can have a unique 16-bit address (am_addr_t) specified on the make command
  - make install.[address] platform

- Two special address constants:
  - TOS_BCAST_ADDR (0xFFFF) is reserved for broadcast traffic
  - TOS_NODE_ID always refers to the node’s own address

- 8-bit group ID to create virtual network/subnetwork

- Each message also has an 8-bit Active Message ID (am_id_t) analogous to TCP ports
  - Determines how host should handle received packets, not which host receives it
  - 0 - 126 are reserved for TinyOS internal use
TinyOS Active Messages (AM)

- `message_t` structure
- Each platform defines platform-specific header, footer, and metadata fields for the `message_t`
- Applications can store up to `TOSH_DATA_LENGTH` bytes payload in the data field (28 by default, 114 max)

```c
typedef nx_struct message_t {
    nx_uint8_t header[sizeof(message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(message_footer_t)];
    nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```
Split-Phase operation

- Many networking commands take a long time (ms) for underlying hardware operations to complete
- TinyOS makes these long-lived operations split-phase
  - Application issues start...() command that returns immediately
  - An event is signaled when it’s actually done

```c
interface SplitControl {
    command error_t start();
    event void startDone(error_t error);

    command error_t stop();
    event void stopDone(error_t error);
}
```

Error code here indicates how TinyOS started processing the request
Error code here indicates how TinyOS completed processing the request
Active Message Interface

interface AMSend {
    command error_t send(am_addr_t addr, message_t * msg, uint8_t len);
    event void sendDone(message_t * msg, error_t error);
    command error_t cancel(message_t * msg);
    command uint8_t maxPayloadLength();
    command void* getPayload(message_t * msg, uint8_t len);
}

interface Receive {
    event message_t* receive(message_t * msg, void * payload, uint8_t len);
}

send is a split-phase operation

Fired on another mote when packet arrives
interface Packet {
    command void clear(message_t * msg);
    command void* getPayload(message_t * msg, uint8_t len);
    command uint8_t payloadLength(message_t * msg);
    command void setPayloadLength(message_t * msg, uint8_t len);
    command uint8_t maxPayloadLength();
}
Other networking interfaces

- Default behavior: no ACKs
- Even with ACKs enabled, no automatic retransmissions

```c
interface PacketAcknowledgements {
    async command error_t requestAck(message_t* msg);
    async command error_t noAck(message_t* msg);
    async command bool wasAked(message_t* msg);
}
```
Multi-Hop Routing

- Collection Tree Protocol (CTP)
  - Link estimator (based on ETX), routing engine, forwarding engine

- Dissemination Protocol

- Berkeley Low-Power IP Stack (BLIP) - 6LowPANs
  - IP Routing over 802.15.4
  - Adaptation layer
    - Header compression
    - Fragmentation
802.15.4 Radio Channels

- Use ISM 2.4 GHz band
- Consist of 16 channels (11-26)
- Lead to interference between motes and 802.11, Bluetooth, etc. devices.
Changing Channel

- Default is channel 26
- Command-line: `CC2420_CHANNEL=xx make [platform]`
- makefile: `PFLAGS = -DCC2420_DEF_CHANNEL=xx`
- Run-time:
  - `CC2420ControlIC` component

```c
interface CC2420Config
{
    command uint8_t getChannel();
    command void setChannel(uint8_t channel);
    command error_t sync();
    event void syncDone(error_t error);
    ...
}
```
Obtaining Sensor Data

- Each sensor component provides one or more split-phase Read interfaces

```c
interface Read<val_t> {  
    command error_t read();
    event void readDone(error_t result, val_t val);
}
```

- Some sensor drivers provide additional interfaces for bulk (ReadStream) or low-latency (ReadNow) readings

- Sensor components are stored in:
  - $TOSROOT/tos/platform/[platform]
Sensor Network Architecture

- Sensors
- Base station
- Gateway
- Internet
Mote - PC Communication

- TinyOS apps can send or receive data over the serial/USB connection to/from an attached PC

- The `SerialActiveMessageC` component provides an Active Messaging interface to the serial port:

```plaintext
components SerialActiveMessageC;
MyAppP.SerialAMSend -> SerialActiveMessageC.Send[AM_SENSORREADING];
MyAppP.SerialReceive -> SerialActiveMessageC.Receive[AM_SENSORREADING];
MyAppP.SerialPowerControl -> SerialActiveMessageC;
```
Mote-PC Serial Communication

- Print Raw Packets using Java Listen Tool

```java
java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB0:telosb
```

packet source
Serial Forwarder

- Java SDK connects to SerialForwarder and converts TinyOS messages to/from native Java objects.

  ```java
  java net.tinyos.sf.SerialForwarder -comm
  serial@[port]:[speed]
  ```

- Let’s applications connect to it over a TCP/IP stream in order to use that packet source (serial port)

- **mig** application auto-generates message object from packet description

  ```bash
  mig java -java-classname=[classname]
  [header.h] [message-name] -o [classname].java
  ```
TOSSIM

- Simulate TinyOS applications
- Good way to rapidly test application logic, at the cost of some realism
  - e.g., does not emulate sensing and does not reproduce timing of real microcontrollers
- Besides app code, need two configuration details:
  - Topology of simulated network – e.g. signal strength
  - Noise trace from environment - e.g. ambient noise

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>-90.80</td>
<td>-39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-95.95</td>
<td>-98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>-97.48</td>
<td>-98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-102.10</td>
<td>-98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>-111.33</td>
<td>-99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-115.49</td>
<td>-98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>-104.82</td>
<td>-94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-110.09</td>
<td>-98</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(from 15-15-sparse-mica2-grid.txt) (from meyer-heavy.txt)
Putting it All Together
Demo

- DemoMessage.h
- DemoAppC.nc (configuration)
- DemoP.nc (module)
- Makefile
- JAVA
  - Main.java
  - Makefile

```
COMPONENT=DemoAppC
TINYOS_ROOT_DIR?=../..\include
$(TINYOS_ROOT_DIR)\Makefile.include
```
Define a new message type

```c
#ifndef __DEMOAPP_H
#define __DEMOAPP_H
enum
{
    AM_DEMO_MESSAGE = 150,
};
typedef nx_struct demo_message
{
    uint16_t photoReading;
} demo_message_t;
#endif // __DEMOAPP_H
```
#include "DemoApp.h"
configuration DemoAppC{}
implementation
{
    components DemoP, MainC, new HamamatsuS10871TsrC() as PhotoC;
    components ActiveMessageC;
    components new AMSenderC(AM_DEMO_MESSAGE), new AMReceiverC(AM_DEMO_MESSAGE);
    components LedsC;
    components new TimerMilliC();
    components SerialActiveMessageC as SerialC;

    DemoP.Boot -> MainC;
    DemoP.Photo -> PhotoC;
    DemoP.RadioControl -> ActiveMessageC;
    DemoP.AMSend -> AMSenderC;
    DemoP.Receive -> AMReceiverC;
    DemoP.Packet -> ActiveMessageC;
    DemoP.SerialControl -> SerialC;
    DemoP.SerialAMSend -> SerialC.AMSend[AM_DEMO_MESSAGE];
    DemoP.SerialPacket -> SerialC;
    DemoP.Leds -> LedsC;
    DemoP.Timer -> TimerMilliC;
}
module DemoP
{
  uses interface Boot;
  uses interface Read<uint16_t> as Photo;
  uses interface SplitControl as RadioControl;
  uses interface AMSend;
  uses interface Receive;
  uses interface Packet;
  uses interface SplitControl as SerialControl;
  uses interface Packet as SerialPacket;
  uses interface AMSend as SerialAMSend;
  uses interface Leds;
  uses interface Timer<TMilli>;
}
implementation
{
  message_t buf;
  message_t *receivedBuf;
  task void readSensor();
  task void sendPacket();
  task void sendSerialPacket();
event void Boot.booted()
{
    call RadioControl.start();
    call SerialControl.start();
}

event void RadioControl.startDone(error_t err)
{
    if(TOS_NODE_ID == 0) // sender
        call Timer.startPeriodic(256);
}

event void Timer.fired()
{
    post readSensor();
}

event void RadioControl.stopDone(error_t err){}

event void SerialControl.startDone(error_t err){}

event void SerialControl.stopDone(error_t err){}
```c
task void readSensor()
{
    if(call Photo.read() != SUCCESS)
        post readSensor();
}

event void Photo.readDone(error_t err, uint16_t value)
{
    if(err != SUCCESS)
        post readSensor();
    else
    {
        demo_message_t * payload = (demo_message_t *)call
            Packet.getPayload(&buf,sizeof(demo_message_t));

        payload->photoReading = value;
        post sendPacket();
    }
}
```
task

```c
void sendPacket()
{
    if (call AMSend.send(AM_BROADCAST_ADDR, &buf, sizeof(demo_message_t)) != SUCCESS)
        post sendPacket();
}
```

event

```c
void AMSend.sendDone(message_t * msg, error_t err)
{
    if (err != SUCCESS)
        post sendPacket();
}
```
event message_t * Receive.receive(message_t * msg, void * payload, uint8_t len)
{
    demo_message_t * demoPayload = (demo_message_t *)payload;
    call Leds.set(demoPayload->photoReading / 200);
    receivedBuf = msg;
    post sendSerialPacket();
    return msg;
}
task void sendSerialPacket()
{
    if(call SerialAMSend.send(AM_BROADCAST_ADDR, receivedBuf, sizeof(demo_message_t))!= SUCCESS)
    post sendSerialPacket();
}
event void SerialAMSend.sendDone(message_t* ptr, error_t success)
{
    if(success!=SUCCESS)
    post sendSerialPacket();
}
# Java Makefile

- **mig** command auto-generates DemoAppMsg class

```makefile
BUILD_EXTRA_DEPS = Main.class

Main.class: DemoAppMsg.java
    javac *.java

DemoAppMsg.java: ../DemoApp.h
    nescc-mig java -java-classname=DemoAppMsg ../DemoApp.h
demo_message -o @$

clean:
    rm -f DemoAppMsg.java  *.class
```
import java.io.*;
import net.tinyos.message.*;

public class Main implements MessageListener
{
    MoteIF mote;
    PrintStream outputFile = null;

    public Main()
    {
        try {
            mote = new MoteIF();
            mote.registerListener(new DemoAppMsg(), this);
        } catch (Exception e) {} 
    }

    public void messageReceived(int dest, Message m)
    {
        DemoAppMsg msg = (DemoAppMsg)m;
        String output = (msg.get_photoReading());
        System.out.println("reading: " +output);
    }
    ...
}