|  |
| --- |
|  |
| Wunderbot User Manual |
| **Elizabethtown College** |
|  |
| Robotics and Machine Intelligence Club |
|  |

**Created:** December 4, 2011

**Last Revised:** December 10, 2011

Contents

[1 Java Tools and Libraries 4](#_Toc311306652)

[1.1 Java Setup Notes 4](#_Toc311306653)

[1.2 Eclipse IDE Setup Notes 4](#_Toc311306654)

[1.3 Serial/Parallel Comm Port Java Libraries 5](#_Toc311306655)

[1.3.1 RXTX 5](#_Toc311306656)

[1.4 Kinect Java Libraries 6](#_Toc311306657)

[1.4.1 JKinect 6](#_Toc311306658)

[2 Motor Controller 7](#_Toc311306659)

[2.1 Communication Settings – COM2 7](#_Toc311306660)

[2.2 Motor Controller – RoboteQ AX2550 7](#_Toc311306661)

[3 Remote Control 8](#_Toc311306662)

[3.1 Communication Settings – COM3 8](#_Toc311306663)

[3.2 Remote Control – Spektrum DX6 8](#_Toc311306664)

[3.2.1 Control Configuration 8](#_Toc311306665)

[3.3 Remote Control Receiver – Spektrum AR6000 8](#_Toc311306666)

[3.4 PSoC - Cypress 9](#_Toc311306667)

[4 Wunderbot Java Code Usage 10](#_Toc311306668)

[4.1 cpuStressTest 10](#_Toc311306669)

[4.1.1 RandomMathThread.java 10](#_Toc311306670)

[4.2 keyPressMonitor 11](#_Toc311306671)

[4.2.1 KeyPressMonitor.java 11](#_Toc311306672)

[4.3 kinect 12](#_Toc311306673)

[4.3.1 Kinect.java 12](#_Toc311306674)

[4.4 main 13](#_Toc311306675)

[4.4.1 Main.java 13](#_Toc311306676)

[4.4.2 SerialPortReaderExample.java 14](#_Toc311306677)

[4.5 motorControls 15](#_Toc311306678)

[4.5.1 MotorControllerReader.java 15](#_Toc311306679)

[4.5.2 RS232Watchdog.java 15](#_Toc311306680)

[4.5.3 SpeedController.java 15](#_Toc311306681)

[4.6 remoteControl 16](#_Toc311306682)

[4.6.1 RemoteControlMonitor.java 16](#_Toc311306683)

[4.7 serialPortComm 17](#_Toc311306684)

[4.7.1 SerialPortDataReceiver.java 17](#_Toc311306685)

[4.7.2 SerialPortDataTransmitter.java 18](#_Toc311306686)

[4.7.3 SerialPortInitializer.java 19](#_Toc311306687)

# ****Java Tools and Libraries****

**NOTE:**

* **Some of the Java libraries currently used for the Wunderbot are Windows-specific due to the need for hardware access.**
* **Unless otherwise noted, assume that all installation and configuration notes for Java libraries are for Windows only.**

## Java Setup Notes

* Regardless of whether you are using 32-bit or 64-bit Windows, be sure to download and install the **32-bit** version of JRE (Java Runtime Environment) **version 6 or higher**. This is to ensure compatibility with all Java libraries that the Wunderbot uses.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

## Eclipse IDE Setup Notes

* If you are using 64-bit Windows, be sure to download and install the **64-bit** version of Eclipse.

<http://www.eclipse.org/downloads/>

## Serial/Parallel Comm Port Java Libraries

* Use the open-source RXTX library to access serial and parallel communication ports in Java

### RXTX

* Official Wiki: <http://rxtx.qbang.org/wiki/index.php/Main_Page>
* Here are some good places to get started using the RXTX libraries:
  + <http://rxtx.qbang.org/wiki/index.php/Writing_%22Hello_World%22_to_a_USB_to_serial_converter>
  + <http://rxtx.qbang.org/wiki/index.php/Two_way_communcation_with_the_serial_port>
* (Recommended) If you are using a 32-bit JRE (Java Runtime Environment)
  1. Download the latest stable binary release of the 32-bit RXTX libraries (“rxtx-2.1-7-bins-r2.zip” as of December 2011) from here:

<http://rxtx.qbang.org/wiki/index.php/Download>

* 1. Extract the files to a folder on your computer.
  2. Copy “…\rxtx-2.1-7-bins-r2\RXTXcomm.jar” to:
     + (32-bit Windows) “C:\Program Files\Java\jre7\lib\ext”
     + (64-bit Windows) “C:\Program Files (x86)\Java\jre7\lib\ext”
  3. Copy “…\rxtx-2.1-7-bins-r2\Windows\i368-mingw32\rxtxParallel.dll”

and “…\rxtx-2.1-7-bins-r2\Windows\i368-mingw32\rxtxSerial.dll” to:

* + - (32-bit Windows) “C:\Program Files\Java\jre7\bin”
    - (64-bit Windows) “C:\Program Files (x86)\Java\jre7\bin”
* (Not recommended for the current Wunderbot configuration) If you are using a 64-bit JRE (Java Runtime Environment), download the Windows-x64 RXTX package from here: <http://www.cloudhopper.com/opensource/rxtx/>
  + If you are getting errors when trying to use the RXTX imports such as “import gnu.io.\*;”
    1. Make sure that in your Eclipse Preferences, the 64-bit JRE is being used as the default (found under Preferences > Java > Installed JREs)
       - For example, Location: “C:\Program Files\Java\jre7” instead of “C:\Program Files (x86)\Java\jre7”
       - If necessary, add the 64-bit JRE to the list and set it as default
    2. Make sure the RXTX files (RXTXcomm.jar, rxtxParallel.dll, rxtxSerial.dll) are in the correct folders as listed here: <http://rxtx.qbang.org/wiki/index.php/Writing_%22Hello_World%22_to_a_USB_to_serial_converter#Installing_JavaVM_and_RXTX>

## Kinect Java Libraries

* Use the open-source JKinect library to access the Microsoft Kinect in Java

### JKinect

* Official SourceForge page: <http://sourceforge.net/projects/jkinect/>
* This library provides a Java wrapper for PrimeSense OpenNI’s Kinect driver for Windows
* Features:
  + Allows access to:
    - Color image data
    - Grayscale Image Data
    - Depth image data
* Setting up the Wunderbot JKinect capability on a new computer:
  1. Remove all previous Kinect installations
  2. Go to <http://sourceforge.net/projects/jkinect/> and click the download button to download the latest version of JKinect
  3. Extract the files to a folder on your computer
  4. Go to the “jkinect-0.0.1” folder (“jkinect-0.0.1” as of December 2011, this may change)
  5. Extract KinectInstall.zip to a folder on your computer
  6. Run KinectInstall.exe
* NOTE: The Wunderbot Java project already has the JKinect drivers and libraries included in the project folder and set up in the Eclipse workspace. It is not necessary to make any changes to the project in Eclipse in order to use JKinect on a different computer, as long as the provided Wunderbot Eclipse workspace is used. Just for future reference, below is the process that was done to add JKinect to the Wunderbot Java project in Eclipse:

1. Go to <http://sourceforge.net/projects/jkinect/> and click the download button to download the latest version of JKinect
2. Extract the files to a folder on your computer
3. Copy the contents of “…\jkinect-0.0.1\jkinect-samples\lib” to “…\Wunderbot Eclipse Workspace\Wunderbot\lib”
4. Open Eclipse and open the Wunderbot Eclipse Workspace
5. In Eclipse, right click the “Wunderbot” project folder and select “Properties”
6. In the Properties window, click on “Java Build Path”
7. Click the “Add JARs…” button
8. Browse into “Wunderbot/lib” and use CTRL-Click to select all the .jar files within the “lib” folder and then hit “OK”
9. Hit “OK” again to save and close the Properties window
10. Expand the “Referenced Libraries” in the Wunderbot project
11. Right click “jkinect-0.0.1.jar” and select “Properties”
12. In the Properties window, click on “Native Library”
13. Click the “Workspace…” folder and browse to “Wunderbot/lib”. Select the “dlls” folder and hit “OK”
14. Hit “Apply” and then hit “OK” to save and close the Properties window

# Motor Controller

## Communication Settings – COM2

* COM Port: COM2
* Baud Rate: 9600
* Data Bits: 7
* Start Bits: 1
* Stop Bits: 1
* Parity: Even
* Flow Control: None

## Motor Controller – RoboteQ AX2550

* Configuration:
  + Input Command Mode: 1 = RS232 full duplex, no watchdog
  + Motor Control Mode: 0 = Separate A, B, Speed control, Open loop (default)
  + Amp Limit: 5 = 105A (default)
  + Acceleration: 2 = Medium (default)
  + Input Switch Function: 2 = No action (default)
  + Brake/Coast: N/A
  + Dead-band: N/A
  + Exponentiation on channel 1: 0 = Linear [no exponentiation] (default)
  + Exponentiation on channel 2: 0 = Linear [no exponentiation] (default)
  + Left/Right Adjust: 7 = No adjustment (default)
  + Joystick Calibration: N/A

# Remote Control

## Communication Settings – COM3

* COM Port: COM3
* Baud Rate: 19200
* Data Bits: 8
* Start Bits: 1
* Stop Bits: 1
* Parity: None
* Flow Control: None

## Remote Control – Spektrum DX6

### Control Configuration

* Left Stick
  + Up and Down
    - Scales the throttle speed between 10% and 100% (down being 10% and up being 100%)
* Right Stick
  + Directly Left
    - Rotates the Wunderbot left in its own footprint
  + Directly Right
    - Rotates the Wunderbot right in its own footprint
  + Directly Up
    - Moves the Wunderbot forward
  + Directly Down
    - Moves the Wunderbot backward
  + Up and to the Left
    - Moves the Wunderbot in a forward left turn motion
    - Scales between “directly up” and “up and to the left”
  + Up and to the Right
    - Moves the Wunderbot in a forward right turn motion
    - Scales between “directly up” and “up and to the right”
  + Down and to the Left
    - Moves the Wunderbot in a backward left turn motion
    - Scales between “directly down” and “down and to the left”
  + Down and to the Right
    - Moves the Wunderbot in a backward right turn motion
    - Scales between “directly down” and “down and to the right”

## Remote Control Receiver – Spektrum AR6000

* Required power source: 5V DC connected to “+” and “-“ pins on the “BAT” channel
* Pins and controller correlation:
  + THR = Left stick up and down
  + AILE = Right stick left and right
  + ELE = Right stick up and down

## PSoC - Cypress

* The PSoC is used to read in the digital signals coming from the remote control receiver and converting them into a string of data that is sent through RS232 to the Wunderbot computer
  + Data string format: “$UUUU,VVVV,WWWW,XXXX,YYYY,ZZZZ$”
    - UUUU: Left motor speed percentage (4 bytes of hex)
    - VVVV: Right motor speed percentage (4 bytes of hex)
    - WWWW: Left motor speed direction (4 bytes of hex)
      * 0 = positive speed
      * 1 = negative speed
    - XXXX: Right motor speed direction (4 bytes of hex)
      * 0 = positive speed
      * 1 = negative speed
    - YYYY: Environmental sampler status (4 bytes of hex)
      * 0 = off
      * 1 = on
    - ZZZZ: Emergency stop status (4 bytes of hex)
      * 0 = disabled
      * 1 = enabled
* When using the PSoC along with the remote control receiver with the Wunderbot:
  + GROUND from the remote control receiver **must** be connected to the GROUND on the PSoC
    - The PSoC needs a common reference point with the remote control receiver to be able to properly receive the digital signals from the receiver
  + **Do not** use USB to power both the PSoC and the remote control receiver at the same time
    - There is not enough power to be able to run both devices properly
    - Doing this may cause strange errors that are hard to debug
  + Use a 5V DC power supply to power both the remote control receiver and the PSoC board

# Wunderbot Java Code Usage

## cpuStressTest

### RandomMathThread.java

* Calculates large arrays of random values and performs some math on them in order to put load on the processor
* Starting the same number of these threads as there are processing cores can simulate 100% processor usage
  + Useful for testing the performance of a Java program under near-worst-case conditions

Example code of class usage:

// Create and start running a new random math thread

**new** RandomMathThread**();**

## keyPressMonitor

### KeyPressMonitor.java

* Creates a Java applet that handles any key press or releases when the applet window is in focus
* This class was meant for reading in arrow key presses to control the movement of the Wunderbot
* It is an independently running thread that calls other Wunderbot functions when keys are pressed
* If a different functionality for this class is desired, a modified version of the class must be created

Example code of class usage:

// Create and start a key press monitor applet

KeyPressMonitor**.**startKeyPressMonitor**();**

## kinect

### Kinect.java

* Class for preparing to use a Microsoft Kinect
  + connect()
    - Function for connecting to the Kinect and creating a JKinect object
    - Input:
      * None
    - Output:
      * JKinect object

Example code of function usage:

--This shows how to connect to the kinect and then shows all the possible operations that can be performed with the created JKinect object

// Connect to the Microsoft Kinect

JKinect kinect **=** **new** Kinect**().**connect**();**

// Get Kinect data a few times

**for** **(**int i **=** 0**;** i **<** 3**;** i**++)**

**{**

// Get new data from the Kinect

kinect**.**update**();**

// Get the grayscale image info

byte**[]** grayscaleData **=** kinect**.**getGrayscaleImage**();**

// Get the color image info

byte**[]** colorData **=** kinect**.**getGRGB24Image**();**

// Get the depth image info

int**[]** depthData **=** **new** int**[**640 **\*** 480**];**

depthData **=** kinect**.**getDepth**();**

**}**

// Shutdown the connection to the Kinect

kinect**.**shutDown**();**

## main

### Main.java

* Contains the main() function that gets called when the Wunderbot Java project is run
  + First thread of the Java code that gets created
* All communication, data processing, and/or other desired threads and functionalities must be initiated from this main thread
* This thread is able to exit if it has run through all of its tasks, even if there are still threads running that it created

### SerialPortReaderExample.java

* Example class that shows how you can write a thread that continuously gets a string of serial port input data from the port reader and print it out in the console (or process it however you desire)
* Input: a SerialPortDataReceiver object for access to the thread that is receiving data from the serial port

Full code (for backup and convenience purposes):

package main**;**

**import** serialPortComm**.**SerialPortDataReceiver**;**

public class SerialPortReaderExample **extends** Thread

**{**

SerialPortDataReceiver portReceiver**;**

public SerialPortReaderExample**(**SerialPortDataReceiver receiver**)**

**{**

// Create the thread and name it

**super(**"SerialPortReader"**);**

// Set up the local receiver object based on the one that was passed in

portReceiver **=** receiver**;**

// Start the thread

start**();**

**}**

public void run**()** **{**

**try** **{**

**while** **(** **true** **)** **{**

String message **=** portReceiver**.**getReceivedData**();**

System**.**out**.**println**(**"Got data: " **+** message**);**

**}**

**}**

**catch(** InterruptedException e **)** **{** **}**

**}**

**}**

Example code of class usage:

--Note that testReceiver is a SerialPortDataReceiver object (see SerialPortDataReceiver.java)

// Create and start the serial port reader thread

SerialPortReaderExample serialReader **=** **new** SerialPortReaderExample**(**testReceiver**);**

// Set the serial port reader thread to have a priority slightly less than the port receiver (OPTIONAL)

serialReader**.**setPriority**(**9**);**

## motorControls

### MotorControllerReader.java

* Thread for reading and processing any data that is received from the motor controller

Example code of class usage:

--Note that mcReceiver is a SerialPortDataReceiver object (see SerialPortDataReceiver.java)

// Create and start the motor controller reader thread

MotorControllerReader mcReader **=** **new** MotorControllerReader**(**mcReceiver**);**

// Set the motor controller reader thread to have a priority

// slightly less than the port receiver (OPTIONAL)

mcReader**.**setPriority**(**9**);**

### RS232Watchdog.java

* Thread that keeps feeding the motor controller’s RS232 watchdog
  + You ***must*** create this thread if the motor controller has RS232 watchdog enabled

Example code of class usage:

--Note that mcTransmitter is a SerialPortDataTransmitter object (see SerialPortDataTransmitter.java)

// Start the motor controller watchdog

**new** RS232Watchdog**(**mcTransmitter**);**

### SpeedController.java

* Thread that processes and writes motor speeds to the motor controller
  + It keeps track of desired motor speeds and current motor speeds
  + It has a function that can be called to set desired motor speeds

Example code of class usage:

--Note that mcTransmitter is a SerialPortDataTransmitter object (see SerialPortDataTransmitter.java)

// Create and start the speed controller thread

SpeedController speedController **=** **new** SpeedController**(**mcTransmitter**);**

// Set the speed controller thread to have a priority slightly less than the port transmitter (OPTIONAL)

speedController**.**setPriority**(**9**);**

## remoteControl

### RemoteControlMonitor.java

* Thread for reading and processing any data received from the PSoC remote control signal handler
  + Sends desired motor speeds to a SpeedController thread (see SpeedController.java)
    - Create a SpeedController thread before creating this thread

Example code of class usage:

--Note that rcReceiver is a SerialPortDataReceiver object (see SerialPortDataReceiver.java)

// Create and start the remote control port monitor thread

RemoteControlMonitor remoteControlMonitor **=** **new** RemoteControlMonitor**(**rcReceiver**);**

// Set the remote control monitor thread to have a priority slightly less than the port receiver (OPTIONAL)

remoteControlMonitor**.**setPriority**(**9**);**

## serialPortComm

### SerialPortDataReceiver.java

* Thread for receiving data from the specified serial comm port
  + Builds up a queue of incoming messages
* Messages must be constantly removed from the queue by another thread using the getReceivedData() function (see SerialPortReaderExample.java for an example of this)

Example code of class usage:

--Note that testSerialPort is a SerialPort object (see SerialPortInitializer.java)

// Create and start the serial port receiver thread

SerialPortDataReceiver serialPortReceiver **=** **new** SerialPortInitializer**().**setUpSerialPortReceiver**(**testSerialPort**);**

// Set the serial port receiver thread to have highest priority (OPTIONAL)

serialPortReceiver.setPriority**(**10**);**

### SerialPortDataTransmitter.java

* Thread for transmitting data to the specified serial comm port
  + Builds up a queue of outgoing messages and sends them
* Use the putDataToTransmit() function at any time to send a string through the serial port

Example code of class usage:

--Note that testSerialPort is a SerialPort object (see SerialPortInitializer.java)

// Create and start the serial port transmitter thread

SerialPortDataTransmitter serialPortTransmitter **=** **new** SerialPortInitializer**().**setUpSerialPortTransmitter**(**testSerialPort**);**

// Set the serial port transmitter thread to have highest priority (OPTIONAL)

serialPortTransmitter**.**setPriority**(**10**);**

// Build the string to send through the serial port

String message **=** "This is a test message."**;**

// Send the message through the serial port

**try**

**{**

serialPortTransmitter**.**putDataToTransmit**(**message**);**

**}**

**catch** **(**InterruptedException e**)**

**{**

e**.**printStackTrace**();**

**}**

### SerialPortInitializer.java

* Class for setting up serial comm port communication
  + connectToCommPort()
    - Function for connecting to a specified serial comm port and creating the relevant SerialPort object
    - Input:
      * String portName – the name of the comm port in the format “COMx”, where ‘x’ is the number of the COM port as defined by Windows (can be found in the Windows Device Manager under “Ports (COM & LPT)”)
      * int baudRate – the baud rate
      * int numDataBits – the number of data bits
      * int numStopBits – the number of stop bits
      * int parityMode – the parity mode
      * int flowControlMode – the flow control mode
    - Output:
      * SerialPort object

Example code of function usage:

// Local variables

SerialPort testSerialPort **=** **null;**

// Connect to the serial port

**try**

**{**

testSerialPort **=**

**new** SerialPortInitializer**().**connectToCommPort**(**

"COM2"**,** /\* COM port name \*/

9600**,** /\* Baud rate \*/

SerialPortInitializer**.**DATABITS\_7**,** /\* Number of data bits \*/

SerialPortInitializer**.**STOPBITS\_1**,** /\* Number of stop bits \*/

SerialPortInitializer**.**PARITY\_EVEN**,** /\* Parity mode \*/

SerialPortInitializer**.**FLOWCONTROL\_NONE**);** /\* Flow control mode \*/

**}**

**catch** **(**Exception e**)**

**{**

e**.**printStackTrace**();**

**}**

* + setUpSerialPortReceiver()
    - Function for creating a SerialPortDataReceiver object
    - Input:
      * SerialPort serialPort – the SerialPort object that a receiver should be created for
    - Output:
      * SerialPortDataReceiver object

Example code for function usage: see SerialPortDataReceiver.java

* + setUpSerialPortTransmitter()
    - Function for creating a SerialPortDataTransmitter object
    - Input:
      * SerialPort serialPort – the SerialPort object that a transmitter should be created for
    - Output:
      * SerialPortDataTransmitter object

Example code for function usage: see SerialPortDataTransmitter.java