## Table of Contents

# Annexes

## *Introduction*

In this section, you will read some useful concepts related with this ebook.

1. Getting Started with LeJOS on Linux

## *Getting started with leJOS on Linux*

### Introduction

This section is a prerrequisite for any project which you use LeJOS APIs for NXT Brick, your computer or your mobile phone with J2ME or Android. In previous releases of this ebook, this material was included in some chapter, but with the time, I have thought that this material is better if it is included in the annexes.

To begin any project using LeJOS it is necessay to do the following tasks before starting.

1. Install Ubuntu
2. Install Java
3. Install LeJOS
4. Install LeJOS firmaware in your NXT brick
5. Install Eclipse
6. Install some plugins for Eclipse
    1. Install LeJOS plugin for Eclipse
    2. Install a SVN plugin for eclipse
    3. Install some code analyzers plugins
        1. Install FindBug plugin
        2. Install PMD plugin
        3. Install Checkstyle plugin

In the following pages, you will learn to install all software required to develop a robust project at home, high school, university or research lab.

## Install Ubuntu

Ubuntu is a very stable and easy to use Linux distro and we recommend to use in your projects with LeJOS. Modern computers has the feature to install Operating Systems using Pendrives.

### How to install Ubuntu from a pendrive?

The way to install in your computer Ubuntu using this method is really easy. Download the ISO for Ubuntu 10.04 in Ubuntu's website. Once you have stored that file (ubuntu-10.04-desktop-i386.iso) Download from Unetbootin's website latest release of the software to install ubuntu from a website. Using a pendrive, execute Unetbootin to convert a pendrive into a installer of ubuntu. Unetbootin is able to run run in Linux systems and Windows so you can begin the experince with Ubuntu and LeJOS from both Operating Systems.

Once you have the Ubuntu installer in your pendrive, check BIOS settins in your computer to configurate Boot settings to configurate that you need to boot from a USB drive. Save BIOS settings and reset your computer to begin boot process from your pendrive. In this moment, you will install Ubuntu in a normal way.

### Links of interest

http://www.ubuntu.com/

http://ubuntu-manual.org/?lang=en_GB

http://unetbootin.sourceforge.net/

## Install Java

LeJOS project is a project based on Java Technology so it is necessary to have installed in your computer latest JDK release. If you use a Debian Distro, I suppose that you have experience with APT manager.

To install latest JDK, open a shell window an type:

```
sudo apt-get update
sudo apt-get install sun-java6-jre sun-java6-jdk sun-java6-plugin
```

### Testing your Java JDK

Once you have installed in your system a Java JDK, it is necessary to test it. To test any Java JDK installation, it is necessary to check the following commands:

1. **javac:** This command is used to compile Java programs.
2. **java:** This command is used to execute any Java program compiled into bytecodes.

Use the following example to check it:

```
public class Test {
      public static void main(String[] arg) {
            System.out.print("My Java installation runs nice!\n");
      }
}
```

Compile and test:

```
jabrena@system1:~/Desktop/> javac Test.java
jabrena@system1:~/Desktop/> java Test
My Java installation runs nice!
```

**Note:**

If you have installed OpenJDK and Java6-Sun in the same system, it is possible to set Java JDK with the following command:

```
sudo update-java-alternatives --set java-6-sun
```

## Install LeJOS project

In this section, you will learn how to install leJOS project in a Ubuntu System. This section is a complement for the file README included in any leJOS release.

### The process

The process to install leJOS on your Unbutu system is the following:

1. Step 1: Download and move LeJOS to the right place
2. Step 2: Install some required libraries/utilities to build leJOS in your system
   1. gcc
   2. Libusb

   3. Ant
   4. Bluez
3. Step 3: Configurate some files on Ubuntu
4. Step 4: Test your LeJOS's installation

**Step 1: Download and move LeJOS to the right place**

Download latest LeJOS release from the following URL:

http://lejos.sourceforge.net/

Extract the file and rename the folder **lejos_nxj** into **lejos** and paste the folder in the path: /user/local/ using root privilegies:

```
jabrena@system1: sudo su
root@system1:/usr/local# tar xvf lejos_NXJ_0_9_0beta.tar.gz
```

when you have the folder lejos in that path, update the permissions for the folder bin:

```
root@system1/usr/local/lejos> chmod +x bin/*
```

**Step 2: Install some required libraries/utilities to build leJOS in your system**

To build your LeJOS release located in /usr/local/lejos/ it is necessary to install sobre required libraries. The libraries to install are:

1. C Compiler, gcc
2. Ant
3. Bluez Library
4. Libusb

The commands to install the libraries are:

```
sudo apt-get install blueman
sudo apt-get install ant
sudo apt-get install build-essential
sudo apt-get install libbluetooth-dev
sudo apt-get install libusb-dev
```

Once you have installed that libraries, the following steps are:

1. Build LeJOS project with Ant
2. Stablish some variables in your system
3. Test your compilation
4. Give IO permissions to manage a NXT brick
5. Test leJOS

**Build LeJOS project with Ant**

Once you have installed gcc,lisbusb,bluez and Ant, you could build the project with Ant:

```
jabrena:/usr/local/lejos/build # ant
Buildfile: build.xml

clean:
     [echo] saving existing files to .bak files

libnxt:

clean:

jlibnxt:
       [cc] 1 total files to be compiled.
       [cc] Starting link

make:

jbluez:

clean:

jbluez:
       [cc] 1 total files to be compiled.
       [cc] /usr/local/lejos/src/jbluez/jbluez.c: In function
'Java_lejos_pc_comm_NXTCommBluez_search':
       [cc] /usr/local/lejos/src/jbluez/jbluez.c:61: warning: 'name_str' may be
used uninitialized in this function
       [cc] Starting link

make:

copy.binaries:
     [copy] Copying 1 file to /usr/local/lejos/bin
     [copy] Copying 1 file to /usr/local/lejos/bin

clear:

build:
     [echo] Done.

BUILD SUCCESSFUL
Total time: 1 second
jabrena:/usr/local/lejos/build #
```

**Configurating some variables in the system.**

To use the system, it is necessary to edit some files to use leJOS in any Linux session. You have to paste the following commands in the files: .profile y /etc/profile

```
export NXJ_HOME=/usr/local/lejos
PATH=$PATH:$NXJ_HOME/bin
export LD_LIBRARY_PATH=$NXJ_HOME/bin
```

**USB Permissions**

A NXT brick is a USB device and it is necessary to give permissions to read and write data. If you use the command **lsusb** you will see all usb connected to your system:

```
jabrena:/usr/local# lsusb
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 003: ID 046d:c016 Logitech, Inc. M-UV69a/HP M-UV96 Optical Wheel
Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 0694:0002 Lego Group
Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
jabrena:/usr/local # ls -la /dev/bus/usb/
```

If you observe the previous list, you will see a Lego device connected to a Ubuntu's system:

```
    Bus 001 Device 002: ID 0694:0002 Lego Group
```

A partial solution to give permissions to that device is:

```
sudo chmod a+w /dev/bus/usb/001/002
```

But this alternative will have to be done every time when your connect and disconnect your NXT brick because values change:

```
system1:/dev/bus/usb/001 # ls
001
system1:/dev/bus/usb/001 # ls
001  003
system1:/dev/bus/usb/001 # ls
001
system1:/dev/bus/usb/001 # ls
001  004
system1:/dev/bus/usb/001 #
```

So this kind of solution is not good. To solve the problem it is necessary to create a file in the path: **/etc/udev/rules.d/**

```
system1:/etc/udev/rules.d # ls
10-board.rules              56-sane-backends-autoconfig.rules
10-vboxdrv.rules            60-pcmcia.rules
```

```
40-alsa.rules                  65-wacom.rules
40-bluetooth.rules             70-kpartx.rules
41-soundfont.rules             70-persistent-cd.rules
51-lirc.rules                  70-persistent-net.rules
51-packagekit-firmware.rules   71-multipath.rules
52-irda.rules                  77-network.rules
55-hpmud.rules                 79-yast2-drivers.rules
55-libsane.rules               99-pcsc_lite.rules
56-idedma.rules
```

Create a file named **70-lego.rules** with the following content:

```
# Lego NXT
BUS=="usb", SYSFS{idVendor}=="03eb", GROUP="lego", MODE="0660"
BUS=="usb", SYSFS{idVendor}=="0694", GROUP="lego", MODE="0660"
BUS=="usb", SYSFS{idVendor}=="0694", GROUP="lego", MODE="0660"
```

Besides, create a group named lego

```
groupadd lego
```

and add all linux users who will use NXT brick in the group lego.

**Note:**

In my case I added the user jabrena to the group lego

**Reboot**

Once you have done previous tasks, reboot your system to test your LeJOS's installation

**Testing your configuration**

If you have done previous tasks, now you can generate nxj files for you NXT brick and send programs to your NXT brick using leJOS technology.

To know if your LeJOS's installation will run nice every time and you have installed and configurated all software dependencies, it is necessary to check your instllation in 3 ways:

1. USB permissions's tests
2. Bluetooth tests
3. Command tests

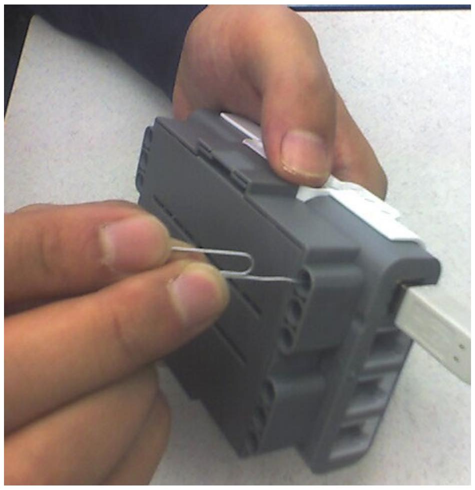
**USB permissions's test**

When you finish your installation, it is necessary to change your firmware on your NXT brick. This action is neccesary to do every time that you reinstall LeJOS with a newer release. So, the first action once you have installed LeJOS, is the action to install latest firmware on your NXT brick. To install LeJOS's firmware, type in a shell nxjflash.

```
nxjflash
```

**Note:**

To install every firmware developed for Lego Mindstorm NXT it is neceesary to put your NXT brick in boot mode. To update the mode in your NXT brick then you have to push reset button. To find that button see at the back of the NXT, upper left corner and push it for more than 5 seconds then you will hear an audibly sound.



If you execute the command nxjflash and you update the firmware on your NXT brick then you have USB permissions ok. In some Ubuntu systems, it is possible that you receive the following exception:

```
“cannot load comm driver”
“an error occurred: Failed to open device in SAM-BA mode.”
```

The solution is:

```
sudo rmmod cdc_acm
sudo -s
echo "" >> /etc/modprobe.d/blacklist.conf
echo "blacklist cdc_adm" >> /etc/modprobe.d/blacklist.conf
```

If you installed the firmware then your NXT brick is ready to execute LeJOS programs.

**Bluetooth tests**

LeJOS offer some utilities to communicate with your NXT brick using a Bluetooth link. The most easier way to test if your Bluetooth capabilities runs nice is using the utility: nxjcontrol. Type nxjcontrol in a Shell to run that utility and try to discover NXT brick with Bluetooth features enabled.

```
nxjcontrol
```

If you don't see any exception in your shell and nxjcontrol detects your NXT brick with LeJOS firmware

**Command tests**

The final set of tests is required to verify if normal commands runs ok in your system, so the main test is based on the creation or a simple program to compile and link in your NXT Brick. Create a simple example in a notepad and save the file as HelloWorld.java

```
import lejos.nxt.*;

public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello World");
        Button.waitForPress();
        }
    }
```

Once you have saved the file, execute the following commands:

```
jabrena@system1:~/projects/lejos/examples/HelloWorld> nxjc HelloWorld.java
jabrena@system1:~/projects/lejos/examples/HelloWorld> nxj -r HelloWorld
leJOS NXJ> Linking...
leJOS NXJ> Uploading...
leJOS NXJ> Linking...
leJOS NXJ> Uploading...
Found NXT: GPSNXT 0016530050A6
leJOS NXJ> Connected to GPSNXT
leJOS NXJ> Upload successful in 744 milliseconds
```

If you execute with success all test then your LeJOS installation is ok.

## Install Eclipse

Develop any Java program in general is easy, but it is more confortable if you use a development tool. In the java market, many developers in the world use Eclipse IDE or Netbeans. In this section I will explain how to develop with Eclipse and the plugin for eclipse.

Eclipse is an extensible, open source IDE (integrated development environment). The project was originally launched in November 2001, when IBM donated $40 million worth of source code from Websphere Studio Workbench and formed the Eclipse Consortium to manage the continued development of the tool.

The stated goals of Eclipse are "to develop a robust, full-featured, commercial-quality industry platform for the development of highly integrated tools." To that end, the Eclipse Consortium has been focused on three major projects:

1. The Eclipse Project is responsible for developing the Eclipse IDE workbench (the "platform" for hosting Eclipse tools), the Java Development Tools (JDT), and the Plug-In Development Environment (PDE) used to extend the platform.
2. The Eclipse Tools Project is focused on creating best-of-breed tools for the Eclipse platform. Current subprojects include a Cobol IDE, a C/C++ IDE, and an EMF modeling tool.
3. The Eclipse Technology Project focuses on technology research, incubation, and education

using the Eclipse platform.

Download Eclipse Europa 3.3 Classic from eclipse’s website. Use the following URL: http://www.eclipse.org/downloads/

## Install Eclipse plugins

Once you have the basic elements installed in your system (Java, LeJOS & Eclipse), you could some features to your Eclipse environment to be more confortable when you develop your projects.

The plugins reviewed are:

1. LeJOS plugin
2. SVN plugin
3. FindBugs plugin
4. PMD plugin
5. CheckStyle plugin

### Eclipse plugin for leJOS

When you execute Eclipse and you want to install any Eclipse plug-in, in this case NXJ Plug-in, you have to go to help > software updates > find and install:

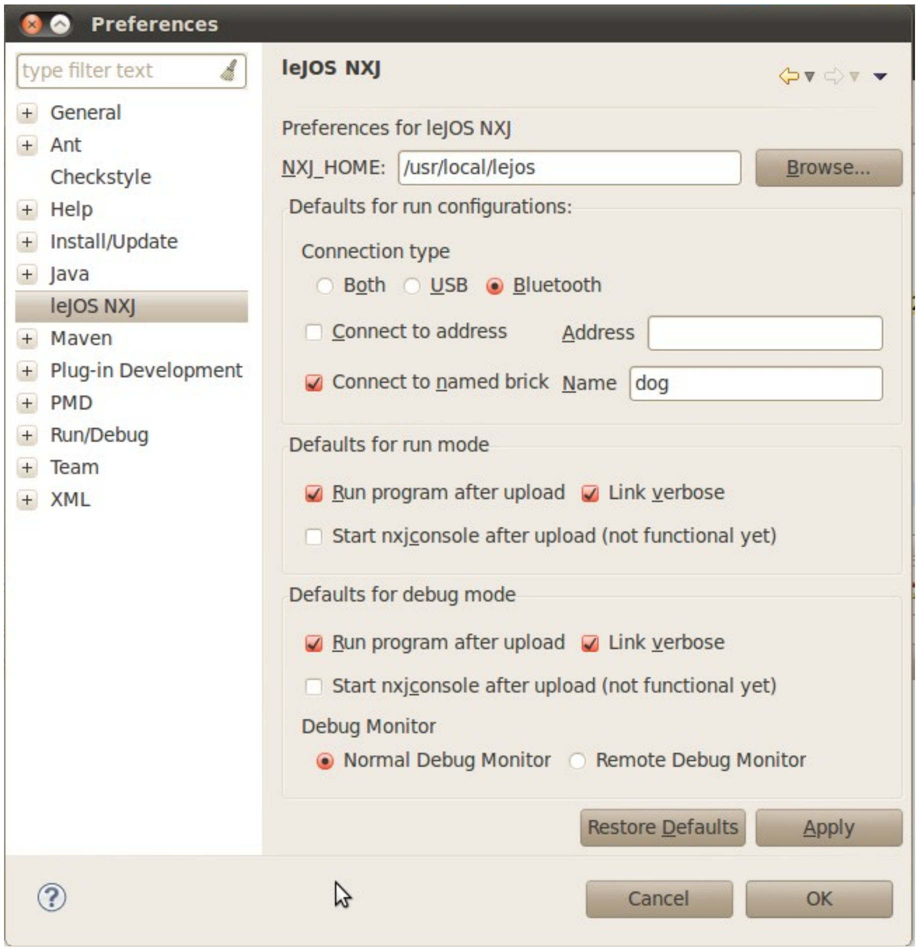
The parameters to write in the next window are:

Name: leJOS NXJ

URL: http://lejos.sourceforge.net/tools/eclipse/plugin/nxj/

### Configuring Eclipse plugin for leJOS

Once you have installed NXJ Plug-in, you have to set NXJ_HOME variable in preference area in Eclipse.



leJOS NXJ

Preferences for leJOS NXJ

NXJ_HOME: /usr/local/lejos

Defaults for run configurations:

Connection type

Both USB [✓ Bluetooth]

☐ Connect to address Address

☑ Connect to named brick Name dog

Defaults for run mode

☑ Run program after upload ☑ Link verbose

☐ Start nxjconsole after upload (not functional yet)

Defaults for debug mode

☑ Run program after upload ☑ Link verbose

☐ Start nxjconsole after upload (not functional yet)

Debug Monitor

[✓ Normal Debug Monitor] Remote Debug Monitor

**NXJ Plugin Features**

NXJ Plug-in allow to NXJ developers to do the following actions:

1. Upload latest firmware into NXT Brick
2. Upload NXJ programs into your NXT Brick
3. Convert any Java project into NXJ project

Besides, NXJ Plugin includes a excellent documentation.

**SVN Plugin for Eclipse**

LeJOS project is a live project and continously is improving so we recommend to install a SVN client to update your local copy of the project. One benefit to have this software appear when LeJOS

Developer Team launch an internal release for developers to tests new concepts.

Eclipse IDE has a SVN plugin, subversive. I recommend to use this plugin to manage the code and use RapidSVN to manage other files which they are not Java code.

**Rapid SVN.**

```
  sudo apt-get install rapidsvn
```

**Subversive**

```
sudo apt-get install subversion

sudo apt-get install libsvn-java
```

http://subclipse.tigris.org/

Subclipse - http://subclipse.tigris.org/update

**Note:**

Test RapidSVN to make a checkout from LeJOS project:

https://lejos.svn.sourceforge.net/svnroot/lejos/trunk

http://rapidsvn.tigris.org/

http://www.eclipse.org/subversive/

**Maven plugin for eclipse**

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

http://maven.apache.org/

http://marketplace.eclipse.org/content/maven-integration-eclipse

http://www.sonatype.com/books/mvnref-book/reference/

### Findbug plugin for eclipse

FindBugs is a program to find bugs in Java programs. It looks for instances of "bug patterns" --- code instances that are likely to be errors.

Plugin Name: FindBugs

URL: http://findbugs.cs.umd.edu/eclipse

http://findbugs.sourceforge.net/

http://findbugs.sourceforge.net/manual/index.html

http://marketplace.eclipse.org/content/findbugs-eclipse-plugin

### PMD plugin for eclipse

PMD scans Java source code and looks for potential problems like:

- Possible bugs - empty try/catch/finally/switch statements
- Dead code - unused local variables, parameters and private methods
- Suboptimal code - wasteful String/StringBuffer usage
- Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
- Duplicate code - copied/pasted code means copied/pasted bugs

Plugin Name: PMD

URL: http://pmd.sf.net/eclipse

http://pmd.sourceforge.net/

http://marketplace.eclipse.org/content/pmd-eclipse

### Checkstyle plugin for Eclipse

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard.

Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration file is supplied supporting the Sun Code Conventions. As well, other sample configuration files are supplied for other well known conventions.

Plugin Name: Checkstyle

URL: http://eclipse-cs.sf.net/update/

http://checkstyle.sourceforge.net/

http://marketplace.eclipse.org/content/checkstyle-plug

## Install latest Developer LeJOS firmware

LeJOS is a open source project which launch new releases every year 2 times more less. Meanwhile LeJOS developer team releases continuosly internal releases for developers to test new features, concepts and refactors.

The way to use latest firmware for internal purposes is:

1. Make a backup of your folder /bin/ and /lib/
2. Download a copy of LeJOS repository
3. Find the folder snapshot and copy the folder /bin/ and /lib/ into your leJOS's installation
    1. Remove .bat files from bin folder
4. Test the operation with some LeJOS utility
    1. nxjcontrol
5. Update firmware with the command nxjflash or nxjflashg

## Summary

In this chapter you should be learnt how to Install leJOS in your Linux system. With the time the process should last lest than 1 hour in the process. In my personal opinion, now I prefer to use Linux System than windows system. Linux is more robust and stable than Windows XP. When you have done some instalations, the process last 15 minutes.

## *How to reinstall Lego Firmware*

## Introduction

LeJOS is an excellent platform to develop software for NXT Lego Mindstorm but it is not the unique solutions. If you read the section NXT Programming Software written by Steve Hassenplug in http://www.teamhassenplug.org/NXT/NXTSoftware.html then you will notice that exist several options to develop software in the NXT brick. If you have installed LeJOS firmware and you decided to reinstall Lego firmware, read this section to know how to do.

### Download latest Lego firmware

To reinstall Lego Firmware is necessary to have latest firmware in your computer. Visit http://mindstorms.lego.com/support/updates/ to download the firmware.

### Set your NXT brick in update mode

Once you have stored latest firmware, it is necessary to set your NXT brick in Update mode. To update the mode in your NXT brick then you have to push reset button. To find that button see at the back of the NXT, upper left corner and push it for more than 5 seconds then you will hear an audibly sound.

If you want to be sure, check your Lego Devices connected with your computer then you will see:

### Reinstall Lego firmware

Use Lego Software that you received with your NXT Kit to download Lego Firmware.

Execute the software and search the option Update NXT Firmware in Tools tab.

When you click in that option then you will see an assistant to download the firmware. Select the firmware that you downloaded and click in download button:

When the process finish then you will see all steps with green color and your NXT brick will have Lego Firmware.

## *Working with I2C to develop drivers*

### Introduction

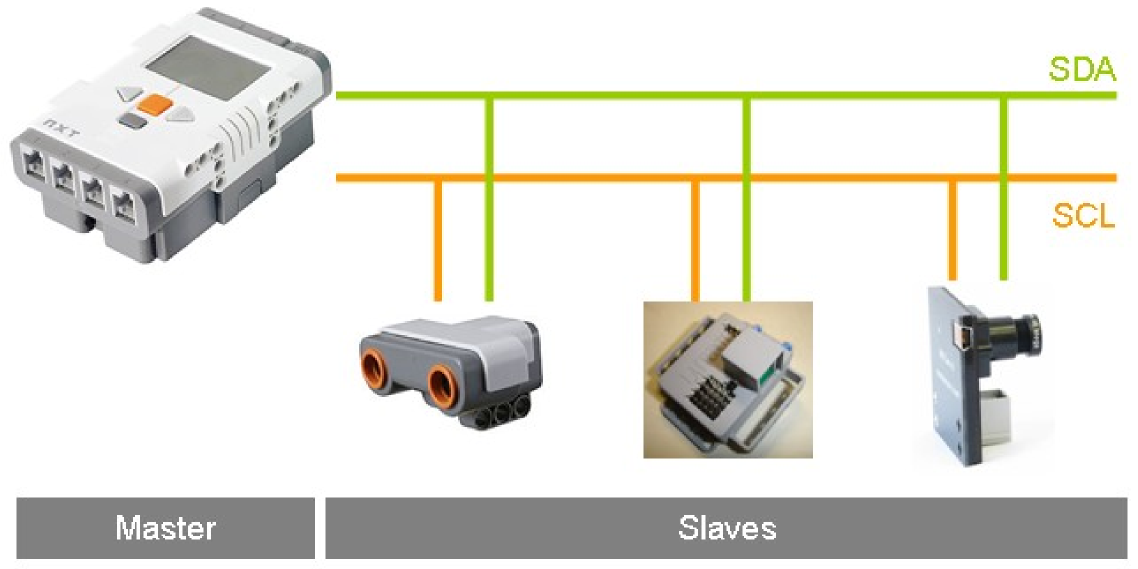
I2C (Inter-Integrated Circuit) is a multi-master serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone. I2C uses only two bidirectional open-drain lines, Serial Data (SDA) and Serial Clock (SCL), pulled up with resistors.

Every device hooked up to the bus has its own unique address.

The bus has two roles for nodes: master and slave:

- Master node: node that issues the clock and addresses slaves
- Slave node: node that receives the clock line and address.

In NXT world the I2C Diagrama should be the following:



There are four potential modes of operation for a given bus device, although most devices only use a single role and its two modes:

- **Master transmit:** master node is sending data to a slave
- **Master receive:** master node is receiving data from a slave
- **Slave transmit:** slave node is sending data to a master
- **Slave receive:** slave node is receiving data from the master

## I2C Bus terminology

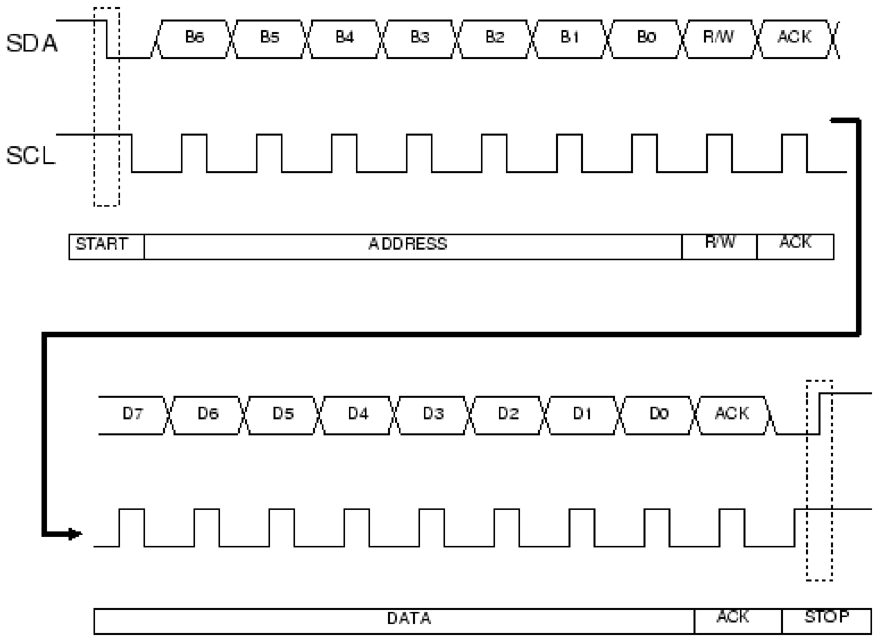
The terminology used when you work with I2C is the following:

- **Transmitter:** The device that sends data to the bus. A transmitter can either be a device that puts data on the bus on its own accord (a 'master-transmitter'), or in response to a request from the master (a 'slave-transmitter').
- **Receiver:** the device that receives data from the bus. A receiver can either be a device that receives data on its own request (a 'master-receiver), or in response to a request from the master (a 'slave-receiver).
- **Master:** the component that initializes a transfer (Start command), generates the clock (SCL) signal and terminates the transfer (Stop command). A master can be either a transmitter or a receiver.
- **Slave:** the device addressed by the master. A slave can be either receiver or transmitter.
- **Multi-master:** the ability for more than one master to co-exist on the bus at the same time without collision or data loss. Typically "bit-banged" software implemented masters are not multi-master capable. Parallel to I²C bus controllers provide an easy way to add a multi-master hardware I²C port to DSPs and ASICs.

- **Arbitration:** the prearranged procedure that authorizes only one master at a time to take control of the bus.
- Synchronization - the prearranged procedure that synchronizes the clock signals provided by two or more masters.
- **SDA:** data signal line (Serial DAta)
- **SCL:** clock signal line (Serial CLock)

**Terminology for bus transfer**

- **F (FREE):** the bus is free or idle; the data line SDA and the SCL clock are both in the high state.
- **S (START) or R (RESTART):** data transfer begins with a Start condition. The level of the SDA data line changes from high to low, while the SCL clock line remains high. When this occurs, the bus becomes 'busy'.
- **C (CHANGE):** while the SCL clock line is low, the data bit to be transferred can be applied to the SDA data line by a transmitter. During this time, SDA may change its state, as long as the SCL line remains low.
- **D (DATA):** a high or low bit of information on the SDA data line is valid during the high level of the SCL clock line. This level must be kept stable during the entire time that the clock remains high to avoid misinterpretation as a Start or Stop condition.
- **P (STOP):** data transfer is terminated by a Stop condition. This occurs when the level on the SDA data line passes from the low state to the high state, while the SCL clock line remains high. When the data transfer has been terminated, the bus is free once again.



## LeJOS API

LeJOS project supports I2C devices connected to NXT brick. Every object which uses I2C protocol inherates from **I2CSensor**

The NXT devices which use I2C are:

- ColorSensor
- CompassSensor
- IRSeeker
- NXTe
- NXTCam
- OpticalDistanceSensor
- PSPNXController
- RCXLink
- RCXMotorMultiplexer
- RCXSensorMultiplexer
- TiltSensor
- UltrasonicSensor

The class I2CSensor has the following I2C methods:

- setAddress
- sendData
- getData

Further information about leJOS API here:
http://lejos.sourceforge.net/nxt/nxj/api/index.html

When it is necessary to write a class to manage a new leJOS device the question to do are:

- What is the I2C address to write and read data?
- What is the list of I2C registers to write and read data?
- How to interpret the values from I2C registers?

## I2C Examples with leJOS

To explain the concepts, I will use a NXT I2C Device, Mindsensors NXTServo. This device has been developed to manage RC Servos.

If we want to read the battery connected to that device, it is necessary to know the following parameters:

1. NXTServo I2C Address: **0xb0**
2. NXTServo I2C Register to read battery level: **0x41**

Now I will write a simple example which read the battery from NXTServo:

```
public class NXTServoTest{

      public static void main(String[] args){
            DebugMessages dm = new DebugMessages();
            dm.setLCDLines(6);
            dm.echo("Testing NXT Servo");

            MSC msc = new MSC(SensorPort.S1);
            msc.addServo(1,"Mindsensors RC Servo 9Gr");

            while(!Button.ESCAPE.isPressed()){
                  dm.echo(msc.getBattery());
            }
            dm.echo("Test finished");
      }
}
```

The class MSC, Mindsensors Servo Controller, manages until 8 RC Servos. I will show 2 internal methods in the class MSC:

**The constructor:**

```
public static final byte NXTSERVO_ADDRESS = (byte)0xb0;

public MSC(SensorPort port){
      super(port);
      port.setType(TYPE_LOWSPEED_9V);
      this.setAddress(MSC.NXTSERVO_ADDRESS);

      this.portConnected = port;
      arrServo = new ArrayList();
}
```

If you observe the code, all I2C operation will use the address 0xb0

**The method getBattery:**

```
public int getBattery(){
      int I2C_Response;
      byte[] bufReadResponse;
      bufReadResponse = new byte[8];
      byte kSc8_Vbatt = 0x41;//The I2C Register to read the battery
      I2C_Response = this.getData(kSc8_Vbatt, bufReadResponse, 1);
      return(37*(0x00FF & bufReadResponse[0]));// 37 is calculated from
      //supply from NXT =4700 mv /128
}
```

In this example we read the I2C register 0x41 which store battery level. Every I2C action has a response. If the response is 0 then it is a success if the result is not 0 then it was a failure. Besides when you read a I2C register, you have to use a buffer, in this case bufReadResponse.

## Migrating code I2C from others platforms

When you develop NXT software, it is a usual that you get ideas from others developers who likes others platforms. In this section I will explain how to migrate I2C RobotC and NXC code.

### Migrating I2C Code from RobotC to Java leJOS

RobotC has the following I2C functions to read and write registers.

| **Function** | **Description** |
|---|---|
| sendI2CMsg(nPort, sendMsg, nReplySize); | Send an I2C message on the specified sensor port. |
| nI2CBytesReady[] | This array contains the number of bytes available from a I2C read on the specified sensor port. |
| readI2CReply(nPort, replyBytes, nBytesToRead); | Retrieve the reply bytes from an I2C message. |
| nI2CStatus[] | Currents status of the selected sensor I2C link. |
| nI2CRetries | This variable allows changing the number of message retries.<br>The default action tries to send every I2C message three times before giving up and reporting an error. Unfortunately, this many retries can easily mask any faults that can exist. |
| SensorType[] | This array is used to configure a sensor for I2C operation. It also indicates whether 'standard' or 'fast' transmission should be used with this sensor. |

Now I will show an example with the same method getBattery:

```
/*===================================
**
** Read the battery voltage data from
** NXTServo module (in mili-volts)
**
===================================*/
int  Get_Batt_V()
{
  byte sc8Msg[5];
  const int kMsgSize       = 0;
  const int kSc8Address    = 1;
  const int kReadAddress   = 2;
      byte replyMsg[2];

  // Build the I2C message
  sc8Msg[kMsgSize]          = 2;
  sc8Msg[kSc8Address]       = kSc8ID ;
  sc8Msg[kReadAddress]      = kSc8_Vbatt ;

  while (nI2CStatus[kSc8Port] == STAT_COMM_PENDING);
  {
   // Wait for I2C bus to be ready
  }
  // when the I2C bus is ready, send the message you built
  sendI2CMsg(kSc8Port, sc8Msg[0], 1);

  while (nI2CStatus[kSc8Port] == STAT_COMM_PENDING);
  {
   // Wait for I2C bus to be ready
  }
  // when the I2C bus is ready, send the message you built
  readI2CReply(kSc8Port, replyMsg[0], 1);

  return(37*(0x00FF & replyMsg[0]));  // 37 is calculated from

      //supply from NXT =4700 mv /128
}
```

Now the migration to Java leJOS:

```
/**
 * Read the battery voltage data from
 * NXTServo module (in mili-volts)
 *
 * @return
 */
public int getBattery(){
      int I2C_Response;
      byte[] bufReadResponse;
      bufReadResponse = new byte[8];
      byte kSc8_Vbatt = 0x41;//The I2C Register to read the battery
      I2C_Response = this.getData(kSc8_Vbatt, bufReadResponse, 1);
      return(37*(0x00FF & bufReadResponse[0]));// 37 is calculated from
      //supply from NXT =4700 mv /128
}
```

## Migrating I2C Code from NXC to Java leJOS

NXC Programming Languaje has a set of functions to manage I2C:

| Function | Description |
|---|---|
| LowspeedWrite(port, returnlen, buffer) | This method starts a transaction to write the bytes contained in the array buffer to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is |

| | |
|---|---|
| | 16. The port may be specified using a constant (e.g., `IN_1`, `IN_2`, `IN_3`, or `IN_4`) or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.<br><br>`x = LowspeedWrite(IN_1, 1, inbuffer);` |
| LowspeedStatus(port, out bytesready) | This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful LowspeedWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer. The port may be specified using a constant (e.g., `IN_1`, `IN_2`, `IN_3`, or `IN_4`) or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.<br><br>If the return value is 0 then the last operation did not cause any errors. Avoid calls to LowspeedRead or LowspeedWrite while LowspeedStatus returns `STAT_COMM_PENDING`.<br><br>`x = LowspeedStatus(IN_1, nRead);` |
| LowspeedCheckStatus(port) | This method checks the status of the I2C communication on the specified port. The port may be specified using a constant (e.g., IN_1, IN_2, IN_3, or IN_4) or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads. If the return value is 0 then the last operation did not cause any errors. Avoid calls to LowspeedRead or LowspeedWrite while LowspeedStatus returns STAT_COMM_PENDING.<br><br>x = LowspeedCheckStatus(IN_1); |
| LowspeedBytesReady(port) | This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful LowspeedWrite call that requested response data from the device then the return value will be the number of bytes in the internal read buffer. The port may be specified using a constant (e.g., IN_1, IN_2, IN_3, or IN_4) or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads. |

| | |
|---|---|
| | x = LowspeedBytesReady(IN_1); |
| LowspeedRead(port, buflen, out buffer) | Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the array buffer provided. The maximum number of bytes that can be written or read is 16. The port may be specified using a constant (e.g., IN_1, IN_2, IN_3, or IN_4) or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads. If the return value is negative then the output buffer will be empty.<br><br>x = LowspeedRead(IN_1, 1, outbuffer); |
| I2CWrite(port, returnlen, buffer) | This is an alias for LowspeedWrite.<br><br>x = I2CWrite(IN_1, 1, inbuffer); |
| I2CStatus(port, out bytesready) | This is an alias for LowspeedStatus.<br><br>x = I2CStatus(IN_1, nRead); |
| I2CCheckStatus(port) | This is an alias for LowspeedCheckStatus.<br><br>x = I2CCheckStatus(IN_1); |
| I2CBytesReady(port) | This is an alias for LowspeedBytesReady.<br><br>x = I2CBytesReady(IN_1); |
| I2CRead(port, buflen, out buffer) | This is an alias for LowspeedRead.<br><br>x = I2CRead(IN_1, 1, outbuffer); |
| I2CBytes(port, inbuf, in/out count, out outbuf) | This method writes the bytes contained in the input buffer (inbuf) to the I2C device on the specified port, checks for the specified number of bytes to be ready for reading, and then tries to read the specified number (count) of bytes from the I2C device into the output buffer (outbuf). The port may be specified using a constant (e.g., IN_1, IN_2, IN_3, or IN_4) or a variable. Returns true or false indicating whether the I2C read process succeeded or failed. This is a higher-level wrapper around the three main I2C functions. It also maintainsa "last good read" buffer and returns values from that buffer if the I2C communication transaction fails.<br><br>x = I2CBytes(IN_4, writebuf, cnt, readbuf); |

To explain the concepts, will show an example from Lattebox NXTe which has leJOS and NXC support:

```
byte  bufConfigureSPI[] = {0x50, 0xF0, 0x0C};

void LowspeedWait()
{
 while(true){
    if (LowspeedCheckStatus(IN_3) == NO_ERR) break;
   }
}

void nxt_init()
{
     ResetSensor(IN_3);
     Wait(100);
     SetSensorType(IN_3, IN_TYPE_LOWSPEED);
     SetSensorMode(IN_3, IN_MODE_RAW);
     ResetSensor(IN_3);
     Wait(100);



     LowspeedWait();
     LowspeedWrite(IN_3,0,bufConfigureSPI);
     LowspeedWait();

}


public static final byte NXTE_ADDRESS = 0x28;
private final byte REGISTER_IIC = (byte)0xF0;//NXTe IIC address

/**
 * Constructor
 *
 * @param port
 */
public NXTe(SensorPort port){
      super(port);

      port.setType(TYPE_LOWSPEED_9V);
      port.setMode(MODE_RAW);

      portConnected = port;

      arrLSC = new ArrayList();

      this.setAddress((int) NXTE_ADDRESS);
      int I2C_Response;
      I2C_Response = this.sendData((int)this.REGISTER_IIC, (byte)0x0c);
}
```

## Summary

In this chapter, you learnt how to use I2C protocol to develop new leJOS classes to suppor new sensors or actuators.

## *How to be a new LeJOS Developer*

### Request a user

In previous section, we see that if you create new code, it is necessary to authenticate then you need a LeJOS user. Write us to get your LeJOS user to collaborate with us.

How to use beta software from leJOS SVN?

If you are an active user then you know that leJOS project releases software every 4-6 months more or less. But in this period, the project suffers many changes and If you want to use latest works into your NXT projects, I recommend you that use the following technique.