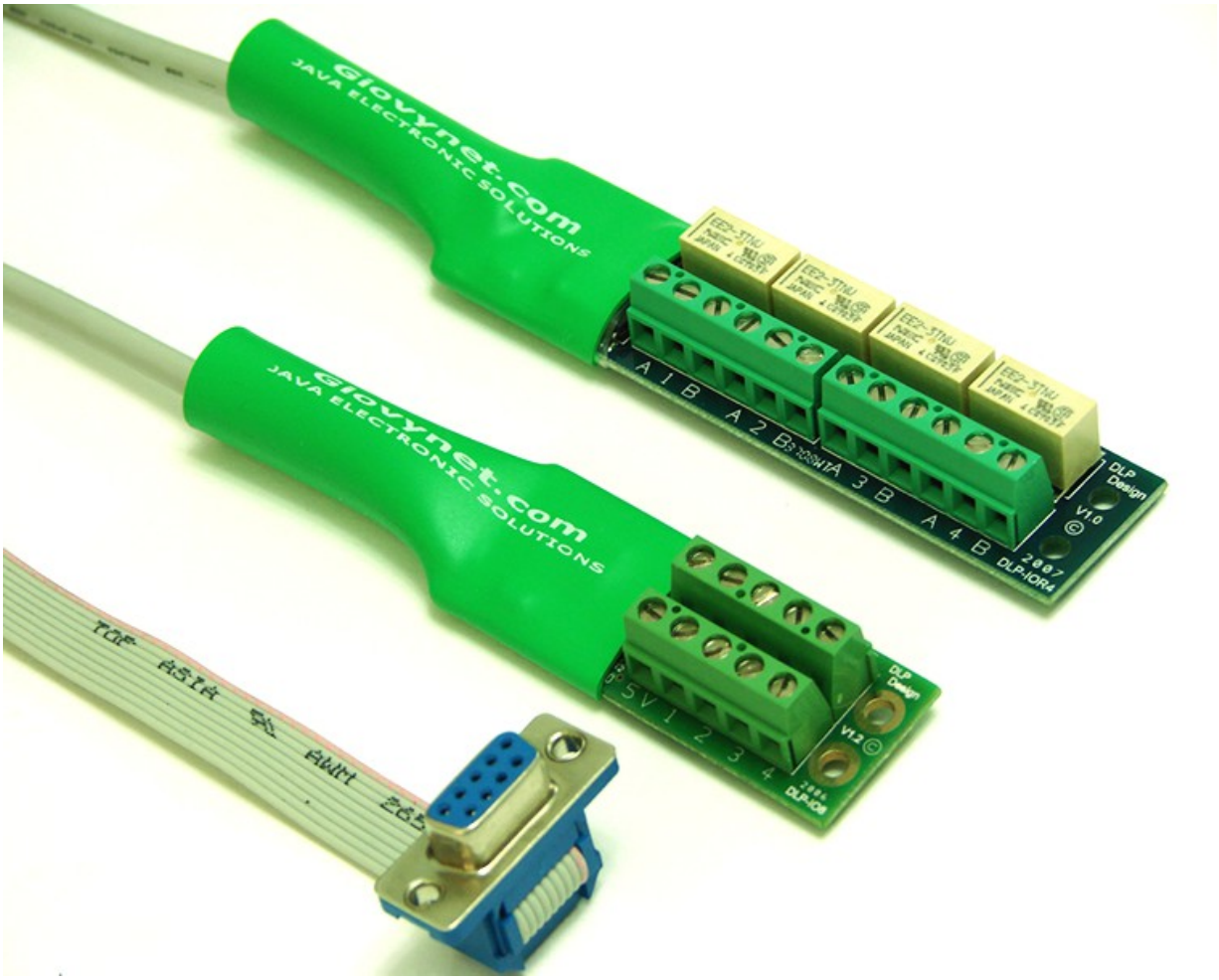
Handbook of Giovynet Driver Version 2.0

Giovanny Rey Cediel



2ª edition

Trademark Acknowledgments

Java, Java Micro Edition (JME), Java Standard Edition (JSE), Java Enterprise Edition (JEE) and Java Runtime Environment (JRE) are trademarks of Oracle Corporation, Inc. DLP-IO8 is a trademark of DLP Design, Inc.

Disclaimer

The information contained herein is subject to change without notice. This document is for informational purposes only. Giovynet.com and its staff make no warranties of any kind for the accuracy, completeness, interpretation or use of the information contained herein. It is your responsibility to comply with all applicable copyright laws. Giovynet.com may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written agreement with Giovynet.com, the furnishing of this document does not grant any license to these patents, trademarks, copyrights or other intellectual property.

Table of Contents

Chapter 1: Introduction to Giovynet Driver

Sometimes the electronic device manufacturers are faced with the problem to communicate or control their creations with a computer, many of them try to work with Java, but decline their intention by the complicated processes that must continue to manipulate external hardware. It is here where Giovynet Driver is presented as an option for Java to handle external circuits.

Formally, Giovynet Driver is a framework that enables the use of Java language to create applications that communicate external circuits with the PC.

Giovynet Driver empowers Java to interface electronic circuits and/or electromechanical circuits with a Java application. Consequently, Java becomes an option for the manufacturer of hardware you want communicate their creations with a PC. In the same way, Giovynet Driver opens the way for Java applications built previously, it is capable of handling mechanisms or machines.

Giovynet Driver version 2.0 uses the following communication ports as a link between Java and the external devices:

- USB port GIV8DAQ: consists of a module which is connected to the computer's USB, no external power supply is required for operation (taken from PC), it has adequate size and design to easily fit an electronic circuit. It has eight independent channels in operations and configurations. Each channel allows Java, to obtain voltage readings, process control, temperature data acquisition, trigger relay, establish logic TTL (0-5VDC), etc.. For technical information about this module please see chapter 4 where there is a detailed discussion about the Java instructions necessary for handling this device.

- USB port GIV4R: consists of a module which is connected to the computer's USB, no external power supply is required for operation (taken from PC), it has adequate size and design to easily fit an electronic circuit. It consists of four independent relays, they keep their state in case of disconnection or shutdown of the PC. This device is designed for users who want to programmatically enable or disable circuits of more power than the PC itself. In its terminals it can resisit voltages up to 220 VDC or 110

VAC with a maximum current of up to 2 Amps. For more details about this module please refer to chapter 5, where there is more discussion about the Java instructions necessary for handling this device.

- **Serial Port:** Giovynet Driver supports sending and receiving ASCII characters via RS-232, in connection with the "null modem" (no handshaking). Only use the lines: transmission, reception and common or "ground" . This indicates that flow control is not performed by "hardware", this task is left to the developer. In chapter 6, the java instructions necessary to manipulate this device will be discussed.

Distributions and Licences Types

Giovynet Driver version 2.0 works under Windows and Linux (tested on Ubuntu/Debian) and is available in three "flavors" depending on usage, number of devices (number of instances), and architecture (x86 or x64). The following describes each of the different distributions of Giovynet Driver:

Giovynet Driver For Personal (x86/x64)

This distribution is free and can be downloaded from the official site of Giovynet.com, it is compiled for x86 (32-bit) or x64 (64-bit). It is designed for developers who wish to make an application for testing and/or training. It allows the use of a single device in an application, this means that you can use either one serial port, or one port GIV8DAQ, or one port GIV4R. If you try to use two or more devices in a application, it will launch an exception. **The license for this distribution prohibits the use of it in work for profit.**

Giovynet Driver For Business: Four Devices (x86/x64)

This distribution can be obtained by a commercial value (see Giovynet.com), you can buy for x86 (32-bit) or x64 (64-bit). It is designed for developers who want to use up to four devices in one application, you can use or instantiate a combination of RS-232, GIV8DAQ, or GIV4R ports. If you are attempting to instantiate more than four devices it will launch an exception to stop the application. **The license for this distribution allows for-profit work.**

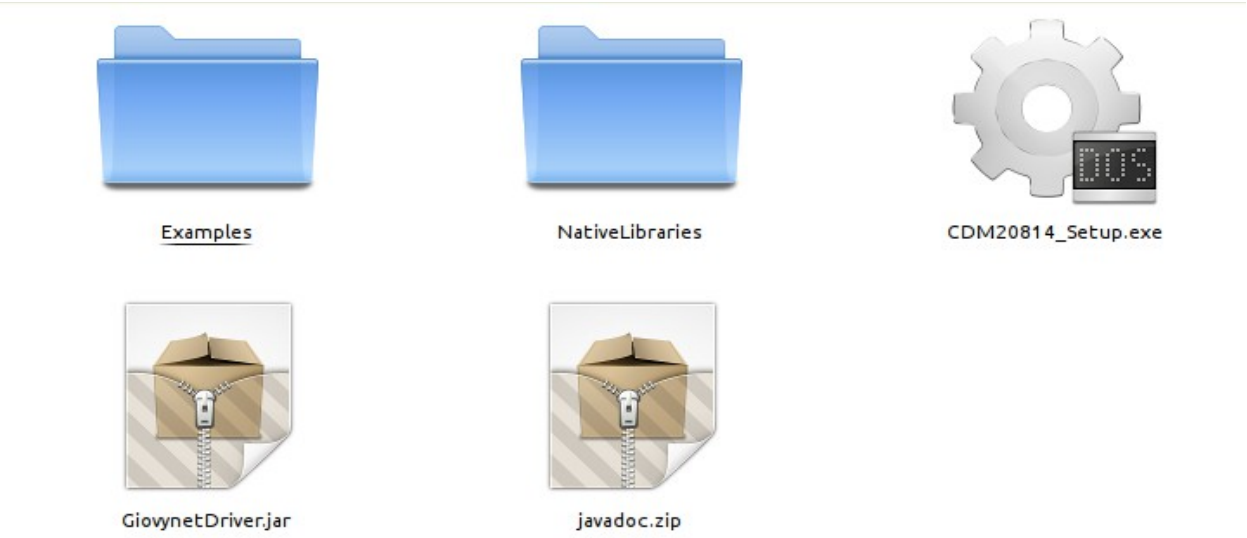
Giovynet Driver For Business: Sixteen Devices (x86/x64)

This distribution has a commercial value (see Giovynet.com for details), you can buy it for x86 (32-bit) or x64 (64-bit). It is designed for developers who want to use up to sixteen devices in an application. You can use or instantiate a combination of RS-232, GIV8DAQ, or GIV4R

ports. If you try to instantiate more than sixteen devices it will cause an exception to stop the application. The license for this distribution allows for-profit work.

Components

Giovynet Driver comes packaged in a zip. When you unzip the file, it generate two folders and three files:



The folder **Examples** contains a project with commented source codes of simple applications whose purpose is to demonstrate the management of communication ports GIV8DAQ, GIV4R, and serial (RS-232).

The folder **NativeLibraries** contains two important files, they are: libSerialPort.dll and libSOSerialPort.so. These files must be in the Java project folder when working on an application, and when you make deployment of the project. These files must be in the same folder where the file is "JAR", which is a result of deployment. Otherwise, the application will launch the exception UnsatisfiedLinkError.

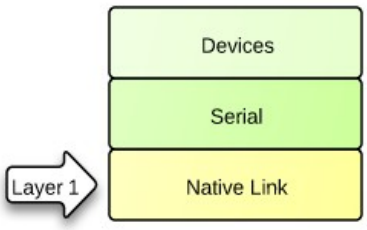
The Java file **GiovynetDriver.jar** contains packages and classes needed to control the physical interfaces (ports GIV, and RS-232) between the external circuit and a Java application. This file must be referenced in the "path" of the project, otherwise the development environment will show syntax errors.

The **javadoc.zip** file is the documentation Java standard of classes and methods of Giovynet Driver.

CDM208014_Setup.exe file consists of a secure and verified application for Windows, it is distributed by Future Technology Devices International Ltd. This application adds "drivers" on Windows for it recognize GIV[1] ports as these ports are built-in with a chip from Future Technology Devices International. Some versions of Windows O.S. do not have this driver included; therefore, it must be installed manually by doubleclicking on this file. This situation does not occurs in Linux because the driver is included in the “kernel” so that Linux recognizes these devices automatically.

Architecture

The internal architecture of Giovynet Driver can be described easily in three layers, where the lower layers support the upper layers.



The first layer, **Native Link**, aims to establish communication between a Java application and operating system. For example, if an application needs to know what ports are connected to the PC, Java sends a query to **Native Link**. When **Native Link** receives the query, it translates it into the language of your operating system and then runs the query. This produces a response that is translated to Java and then sent back to the application.

The second layer, **Serial**, is formed by all Java statements necessary to control the serial ports connected to the PC.

Finally, the third layer **Devices**, is formed by all Java statements necessary to control GIV ports that are the physical interface between the Java application and the external circuit.

1 GIV ports refers to GIV4R and GIV8DAQ ports.

Chapter 2: Installing and Configuring Development Tools

This chapter explains the installation and configuration of Sun JDK and Eclipse IDE for Windows and Linux operating systems. If you are familiar with these issues you can feel free to skip this chapter. Although, the topics addressed here are widely discussed in the community of Java programmers, they are included as a supplementary reference to savvy users or neophytes.

As mentioned in the first chapter, Giovynet Driver is a development framework for Java. Therefore, it requires a set of programs and libraries that let you compile, run and debug Java code. This set of programs and libraries known as Sun JDK (Java Development Kit), is distributed free of charge by Oracle at its official website. There are several versions of Sun JDK for various operating systems amoung which are included Windows and Linux.

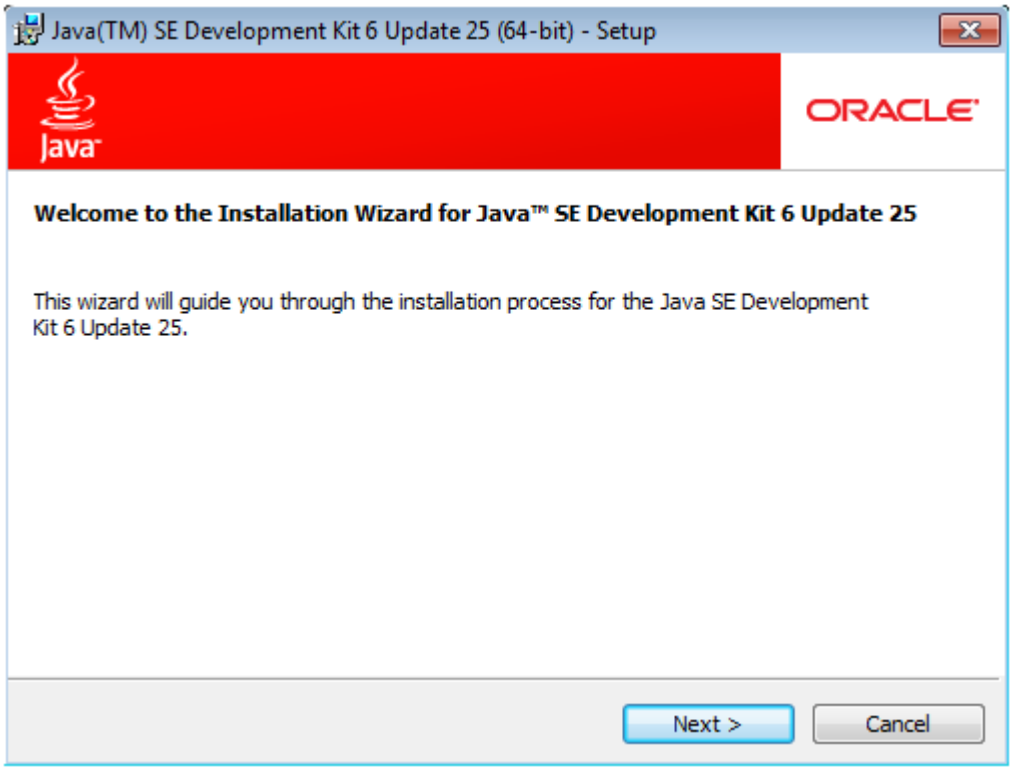
Another tool that is no less important is the Integrated Development Environment or IDE. The IDE is a friendly tool that allows efficient use of the Sun JDK. With an IDE you can build applications quickly and easily.

Currently, there are several Java IDE's to build applications with. The best known are Eclipse and NetBeans; in this book will be used Eclipse IDE.

Installing and Configuring Sun JDK for Windows

The examples cited here were conducted under Windows 7; the procedure varies little in versions like XP or Vista. Installing Sun JDK is easy, first go to the official download site for Oracle: http://www.oracle.com/technetwork/java/javase/downloads/index.html. Select platform and architecture (Windows x32 or Windows x64) and download the file.
The downloaded file will have a name like “jdk-6u25-windows-x.exe,” then double-click on this, it will launch a wizard that guides you step by step in installing the Sun JDK. Upon completion, the system will launch a form to register the product, if desired, this process is free and optional.

The following image shows the installation wizard Sun JDK:



As a result of the installation, it will create a series of folders with distinctive names for each version of Sun JDK. In this case, the image above is created with the path: C:\Program Files\Java\jdk1.6.0_25\bin\, because the JDK version 1.6.025 was installed. In this location, these are all the necessary tools to compile, run and debug your Java source code.

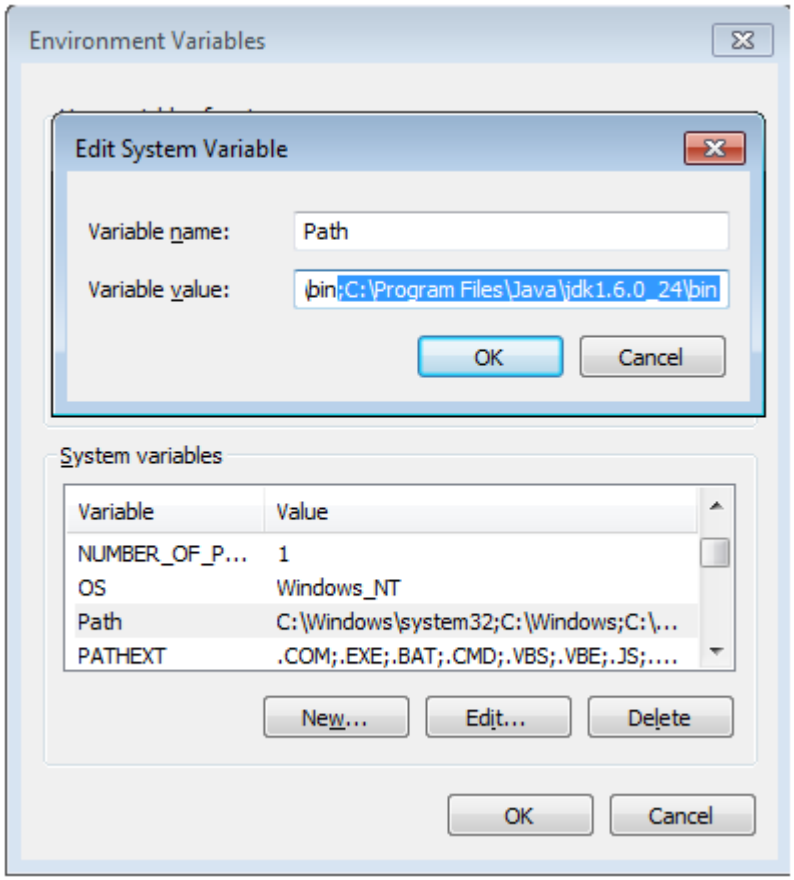
The next step is to configure the installation path in the environment variable path, which will reference the folder jdk1.6.0_25\ bin\, from any path in the system. This is done to invoke the compiler without having to go to the JDK installation path.

To do this:

1. Click on Start.
2. Right click on Computer.
3. Click on Properties.
4. Select the Advanced System Settings.
5. Click on the tab Advanced Settings.
6. Click on the button Environment Variables.

Then, a window appears that shows the system variables. Find the variable "path" in the System Variables. Then, click on the "Edit" button, you will see the editable field with a text string, that represents the values of the variable separated by a semicolon (;), add at the end of this string expression: C:\Program Files\Java\jdk1.6.0_25\bin. This is the path to the binary files unpacked by JDK, then accept and complete.

The following figure shows the editing process of the variable path:



This completes the installation and configuration of the JDK on Windows.

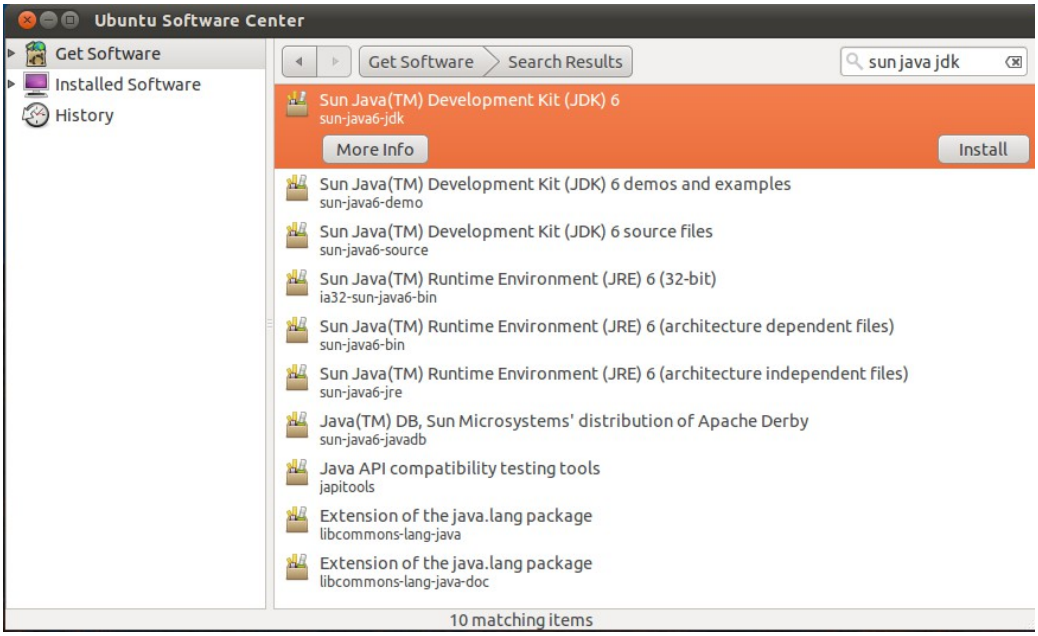
Installing and Configuring Sun JDK for Ubuntu Linux

The examples cited here were conducted under the 11.04 version of Ubuntu Linux. Ubuntu and other versions of Linux included by default, the OpenJDK.

Open JDK implementation is a hundred percent open source Java, the Sun JDK is a Java implementation that is largely open source, but nevertheless it contains some binaries that Oracle has not released licensed open source. Then we will explain how to install Sun JDK.

Open the Ubuntu Software Center, in the search area type "java jdk sun", then you will see a set of results, select the jdk version you want to install (at the time of writing this book the available version is 6).

The following image shows the graphical user interface for Ubuntu Software Center.



Then, press "Install" and wait for the download and installation of Sun JDK to occur.
As a result of the installation, it create a series of folders with a distinctive name for each version of Sun JDK. In this case, the exact version is 1.6.0.24, and the installation path is "/usr/lib/jvm/java-6-sun-1.6.0.24/," this route is important, because there are the libraries that let you compile, run, debug, etc, the Java source code.

To complete this task, you just need to set the environment variables for the Sun JDK. Libraries can be accessed from anywhere on your operating system. This is done by adding the following lines of text at the beginning of the file "/etc/profile" (perform this operation as root):

```
###### Configuration of the environment variables of JAVA ######
JAVA_HOME=”/usr/lib/jvm/java-6-sun-1.6.0.24/”
JRE_HOME=”/usr/lib/jvm/java-6-sun-1.6.0.24/jre/”
CLASSPATH=”.”
PATH=$PATH:/usr/lib/jvm/java-6-sun-1.6.0.24/
export JAVA_HOME
export CLASSPATH
export PATH
```

Then type in the terminal this statement:

```
sudo update-alternatives --config java
```

Then select Sun JDK.

To prove that indeed the configuration was performed successfully, in the same terminal, enter the following statement, and press enter:

```
java -version
```

If everything went as expected, the output of this statement will look something like the following:

```
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
```

Now you are finished with the instalation and configuration of JDK in Ubuntu Linux.

Installing Eclipse IDE for Windows

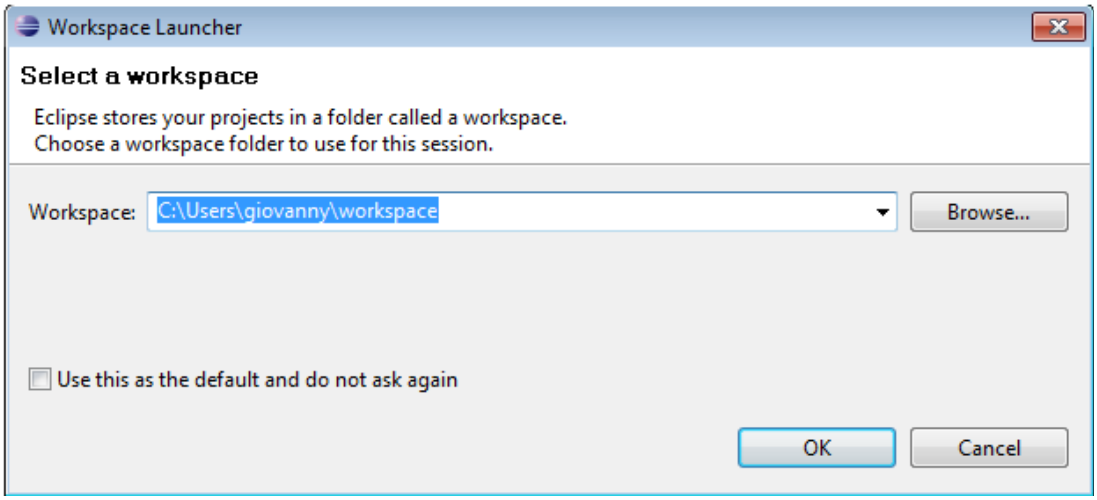
Before you install Eclipse, you must install the Sun JDK. If you have not already done so, please complete it using the steps in "Installing and Configuring Sun JDK for Windows," in this same chapter.

After installing and configuring the Sun JDK, go to the download section on the official site of Eclipse (http://www.eclipse.org/downloads/), seek the distribution "Eclipse IDE for Java Developers", select platform and architecture (Windows x86 or Windows x64) and start to download.

Eclipse IDE is packaged in a "zip" file, unzip this file to produce the folder eclipse. Within this folder, locate the file eclipse.exe and double click on it to start Eclipse. Do this each time you start Eclipse.

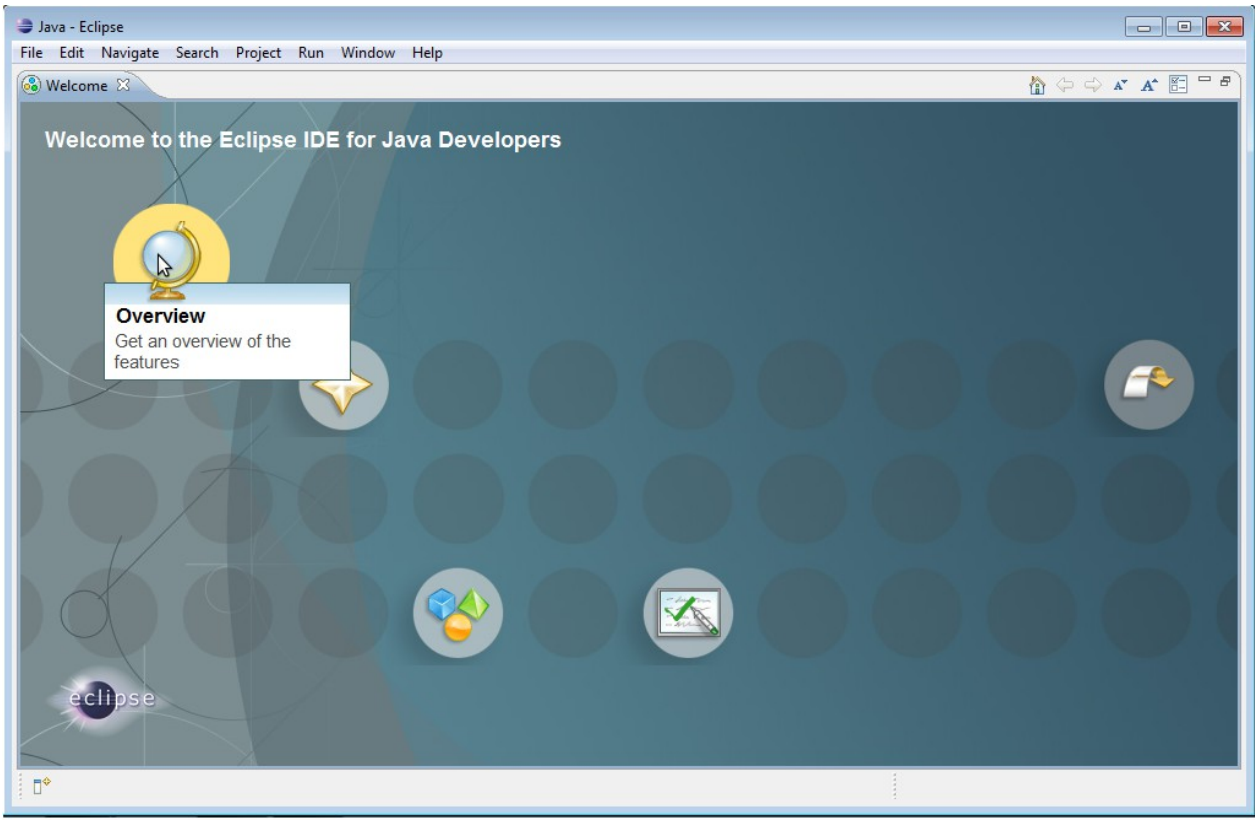
The first thing that appears at the beginning of Eclipse is a wizard to select the location of where to store the project, select a location with permission to create, read and write files. This location is known as Eclipse workspace.

The screenshot below shows the wizard to define the workspace or folder that will store the projects.

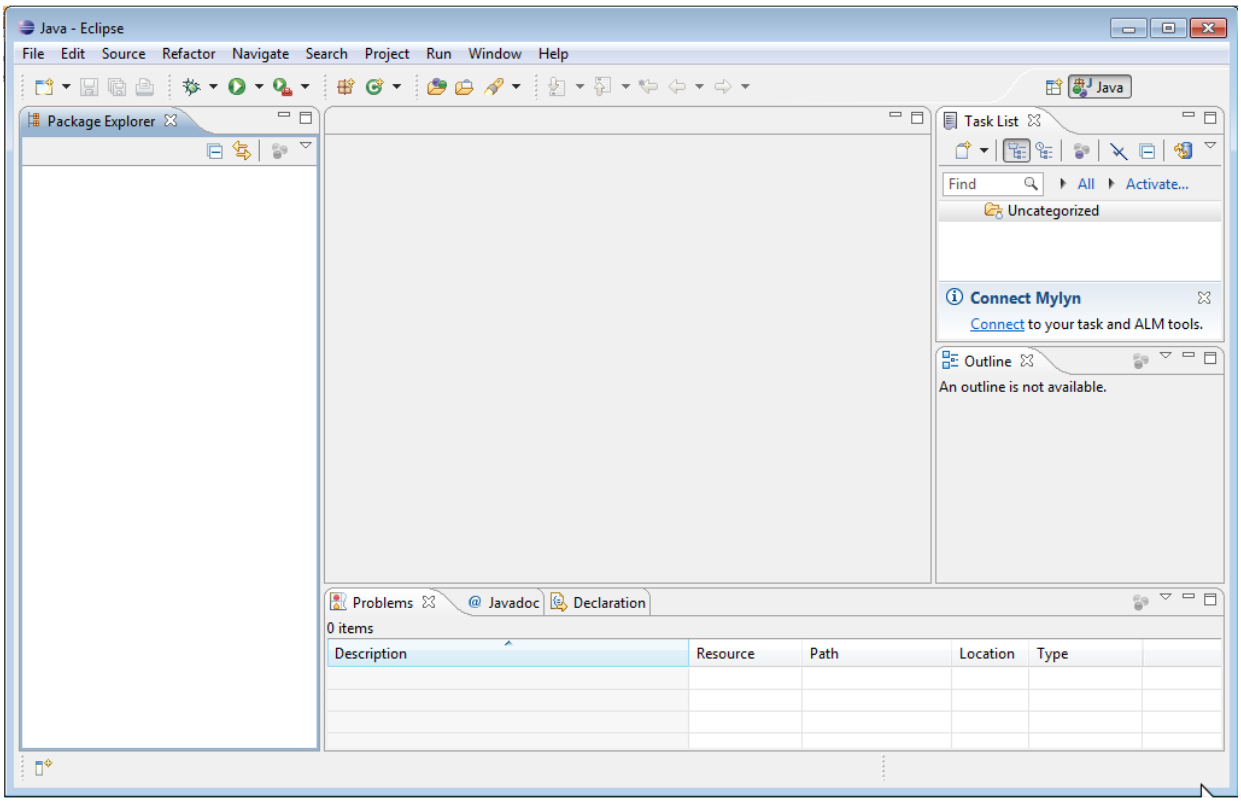


When performing this action for the first time, you will see a tab with a set of links to tutorials to learning Eclipse. You can visit these links if you wish.

The following screenshot shows the welcome tab of Eclipse.



Finally, to go directly to the control panel of Eclipse, click on the "X" in the Welcome tab.
The following figure shows the control panel of Eclipse IDE that appears after closing the Welcome tab



This completes the installation of Eclipse IDE for Windows.

Installing Eclipse IDE for Ubuntu Linux

Before you install Eclipse, you must install the Sun JDK. if you have not already done so, please complete it using the steps described in "Installing and Configuring Sun JDK for Ubuntu Linux," in this same chapter.

After installing and configuring the Sun JDK, go to the download section on the official site of Eclipse (http://www.eclipse.org/downloads/). Locate the distribution "Eclipse IDE for Java Developers", select platform and architecture (x86 or x64 for Linux) and start the download.

Eclipse IDE comes packaged in a "tar.gz." To uncompress it, open a terminal and go to the download file (for example: cd /home/UserFolder/Downloads), then type the statement:

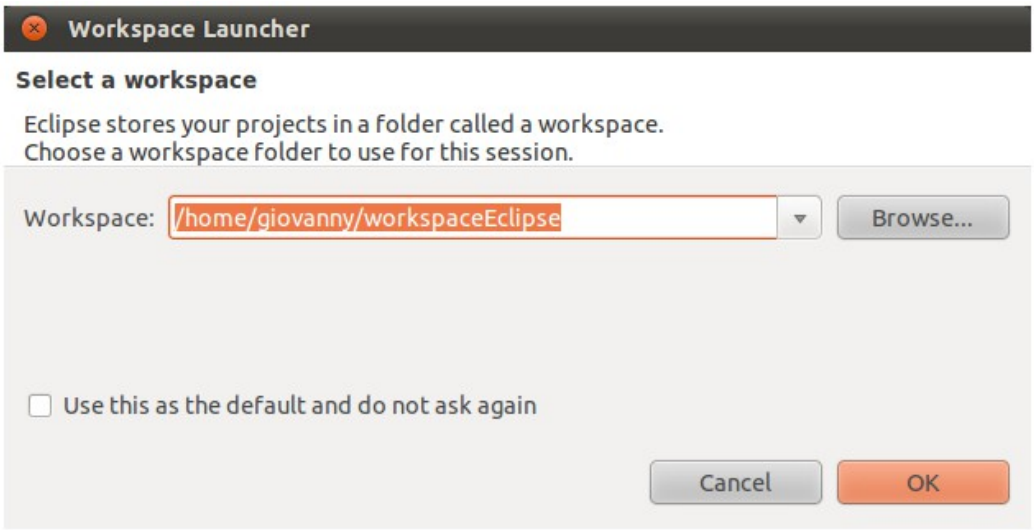
```
sudo tar -xvf FileNameDownloaded.tar.gz
```

Completing this statement will produce a folder named eclipse. To start Eclipse, open a terminal, and then go to the folder created. Run the following statement to start Eclipse:

```
sudo eclipse
```

The first thing that appears when you start Eclipse is a wizard. Select the location where to store the project. Be sure to select a location with permission to create, read and write files. This location is known as Eclipse workspace.

The following screen shows an image of the wizard that defines the workspace or folder that will store the projects.

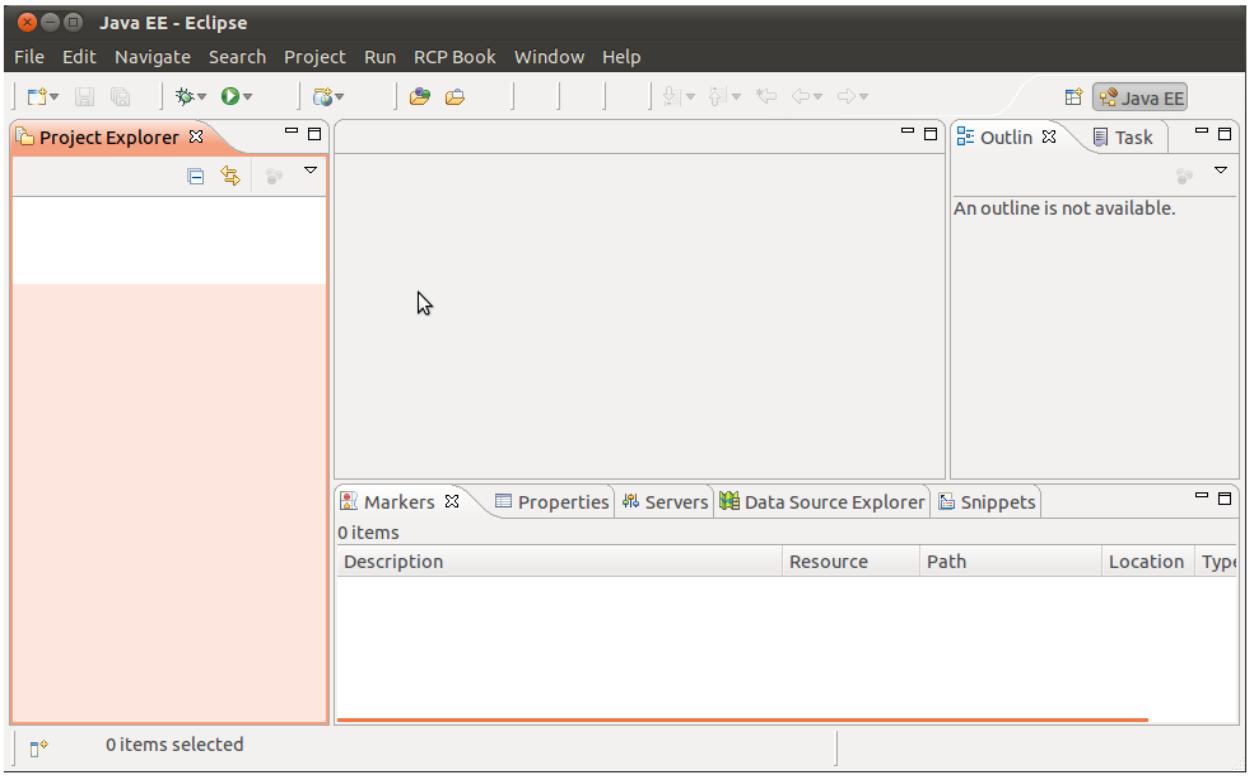


When performing this action for the first time, you will see a tab with a set of links to form a teaching assistant for Eclipse. You can visit these links if you wish.

The following screenshot shows the welcome tab of Eclipse.

Finally, you will need to go directly to the control panel of Eclipse. To do this, click on the "X" in the Welcome tab.

The following figure shows the control panel of Eclipse, which appears after closing the tab welcome.

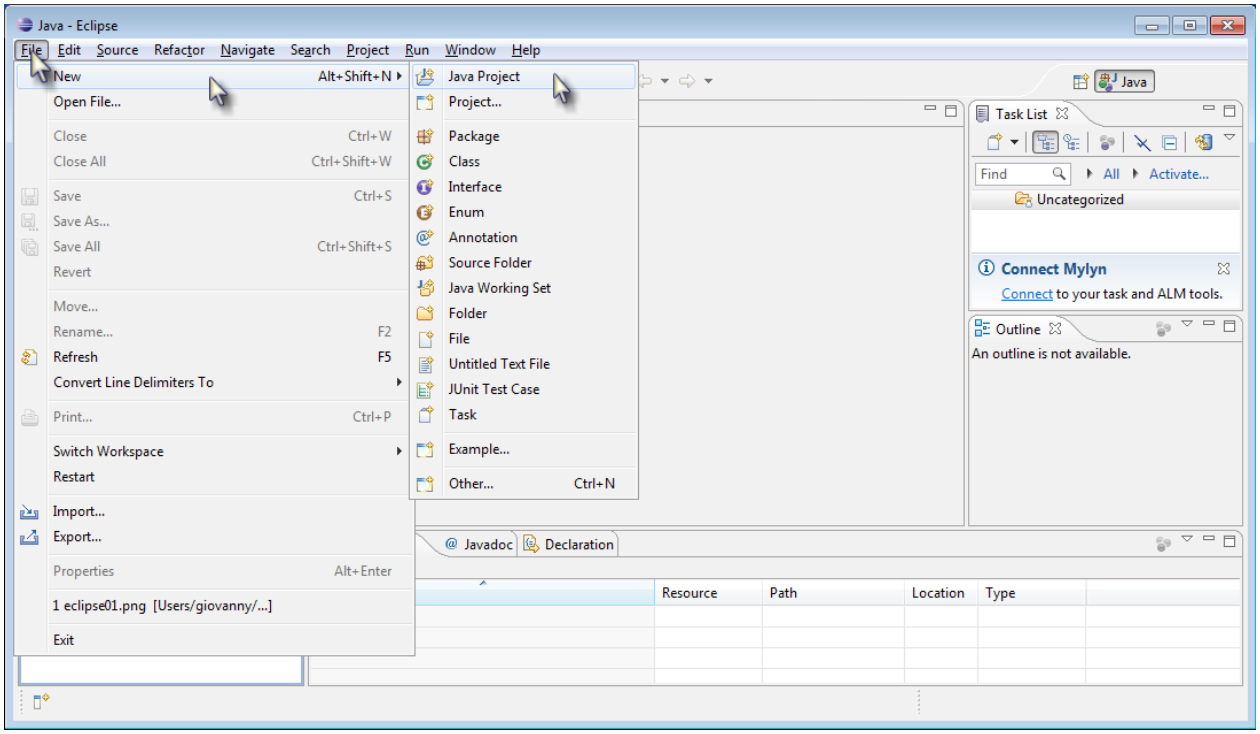


This completes the installation of Eclipse IDE for Ubuntu Linux.

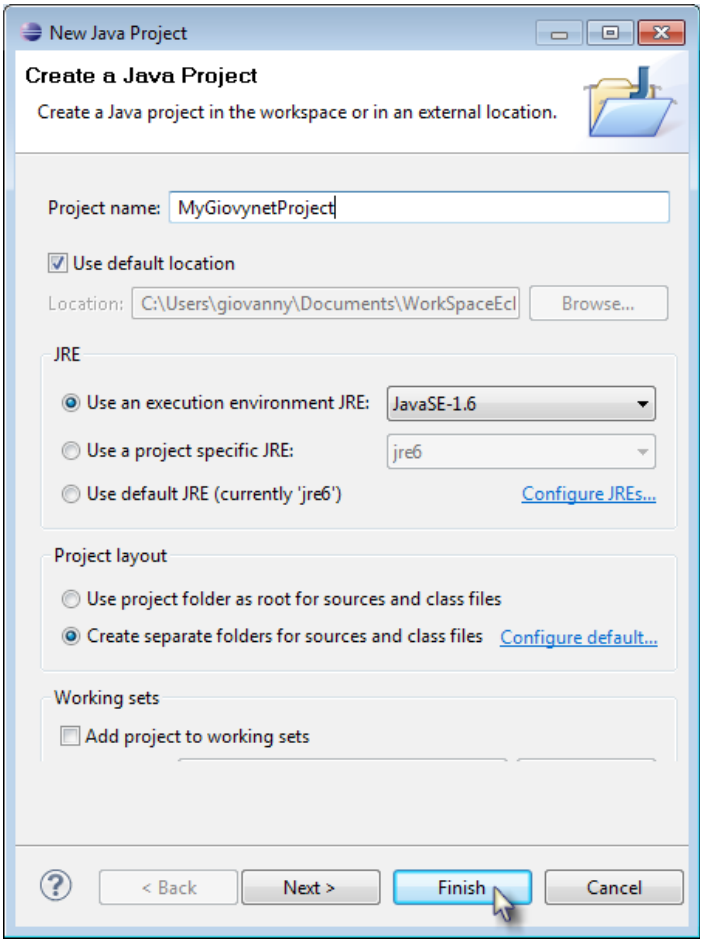
Chapter 3: Creating a Java Project with Giovynet Driver in Eclipse

The first chapter described five components that are part of Giovynet Driver. They only need three components to build a Java and Giovynet Driver project with. Now we will describe how to perform this task.

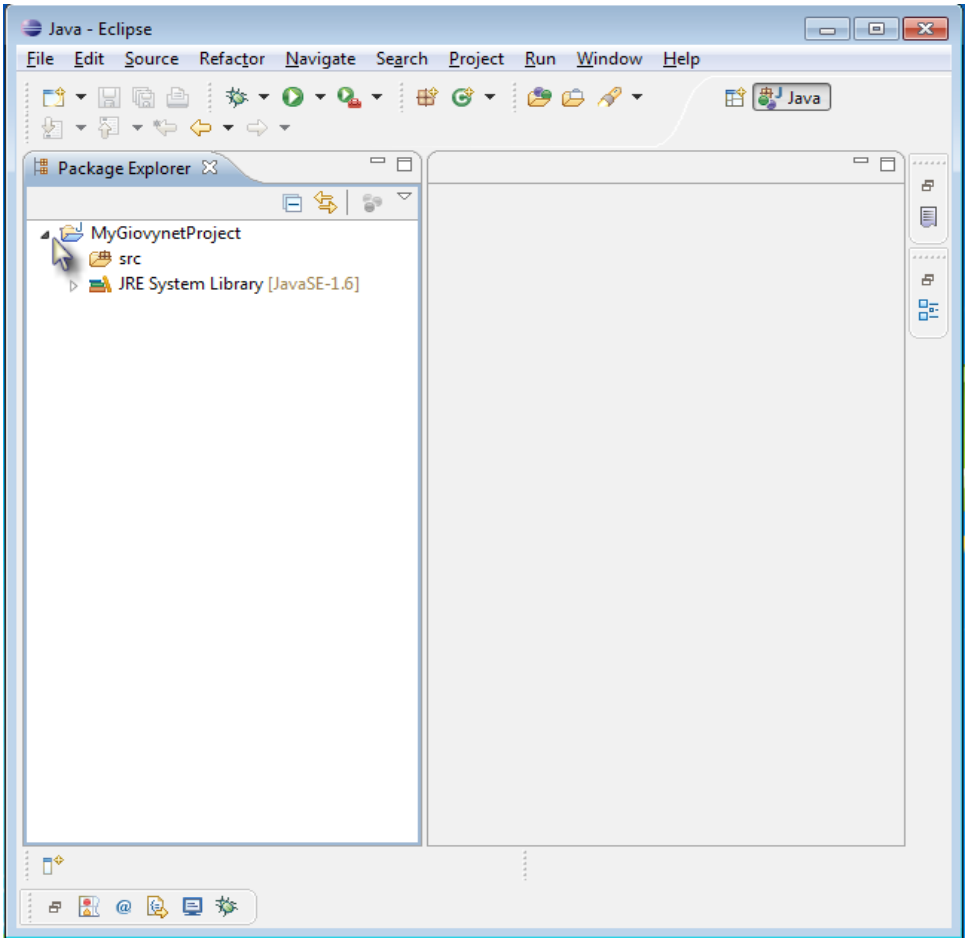
After starting Eclipse, go to the top left of the toolbar and click on File > New > Java Project, as shown in the image below:



Then a wizard will create the project. Enter a descriptive name in the Project Name field and then click on Finish as shown next:



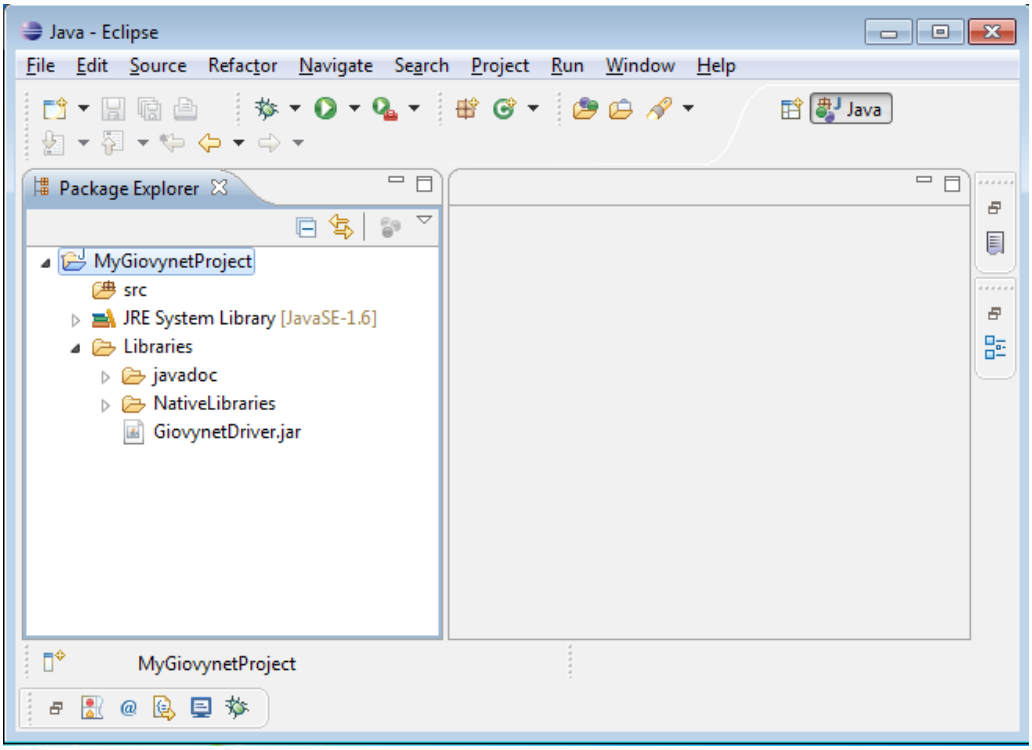
After completing this wizard, you will see a folder with the name of the project in the Package Explorer. The project will be as shown in the image below:



The next step is to create a folder within the project, named Libraries where you add the three components of Giovynet Driver. To do this from Eclipse, right click the folder of the project and select New > Folder. Then you will see a text field where you will type the name of the folder, type the word “Libraries” and then press finish. After this, copy and paste the following components:

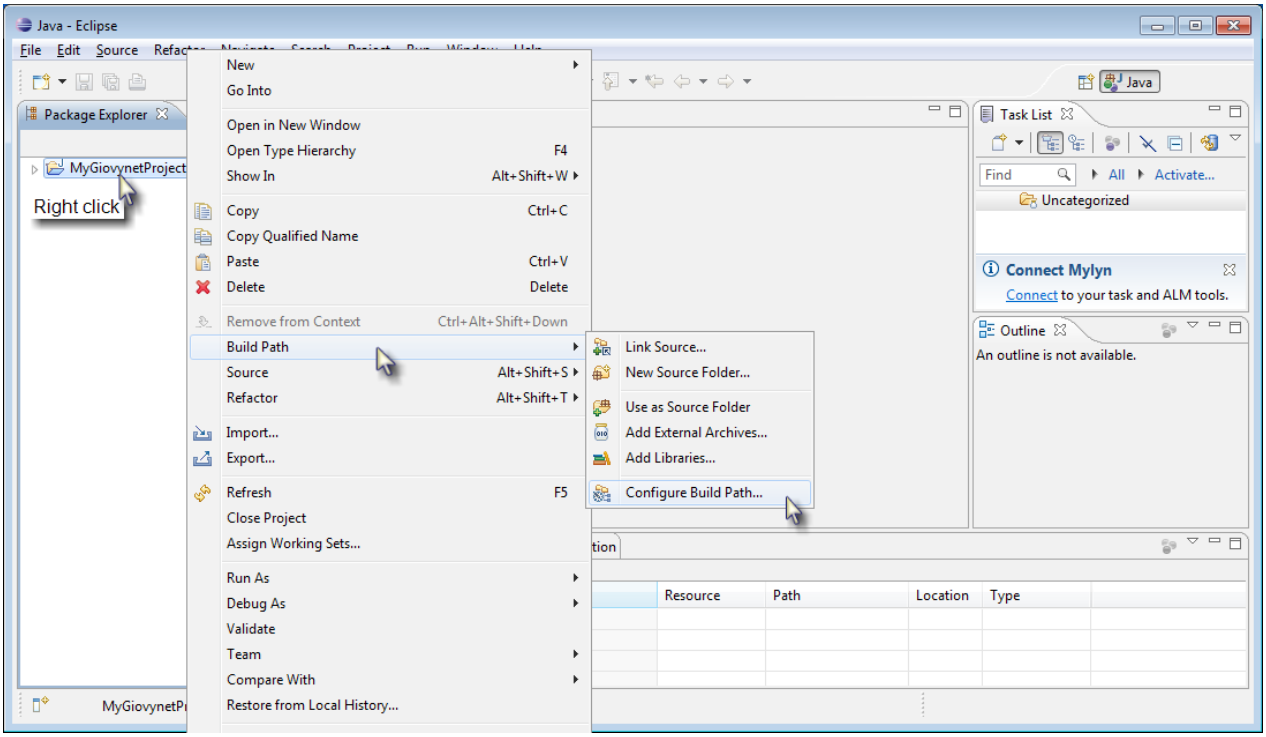
- GiovynetDriver.jar
- NativeLibraries
- javadoc

The project will look like the following picture:

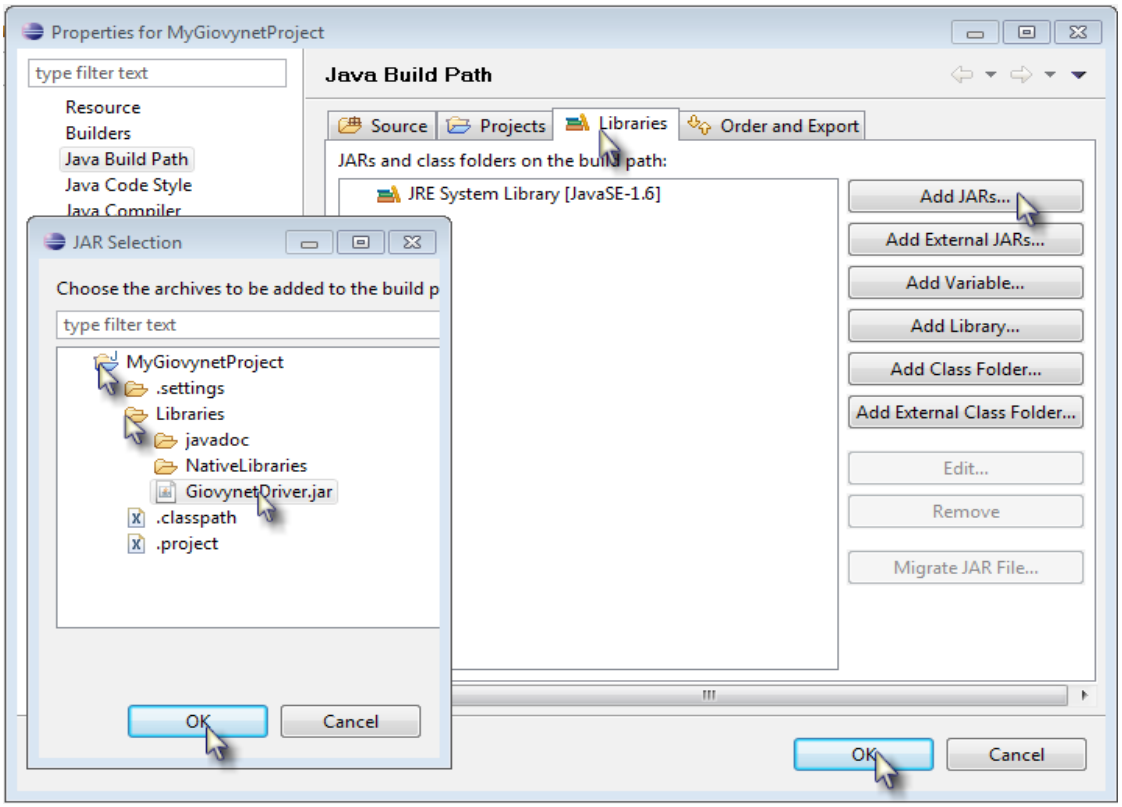


The next step is to link the file Giovynet Driver.jar in the Java project to be able to use the Giovynet Driver routines.

Right click on project name (in the Package Explorer), after select Build Path>Configure Build Path, as shown in the image below:



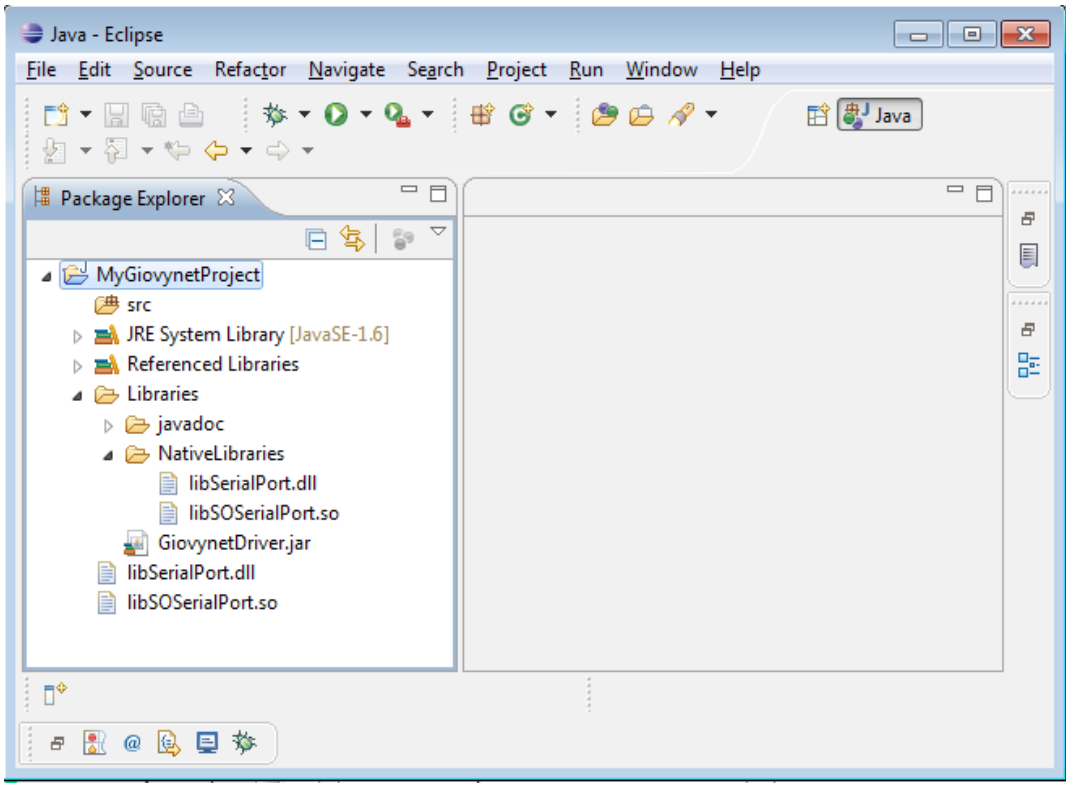
The project properties window appears. Look for the Libraries section, click on the button “Add JARs...” then click on the "Add JARs ..." and select the file GiovynetDriver.jar. As an example consider the following picture:

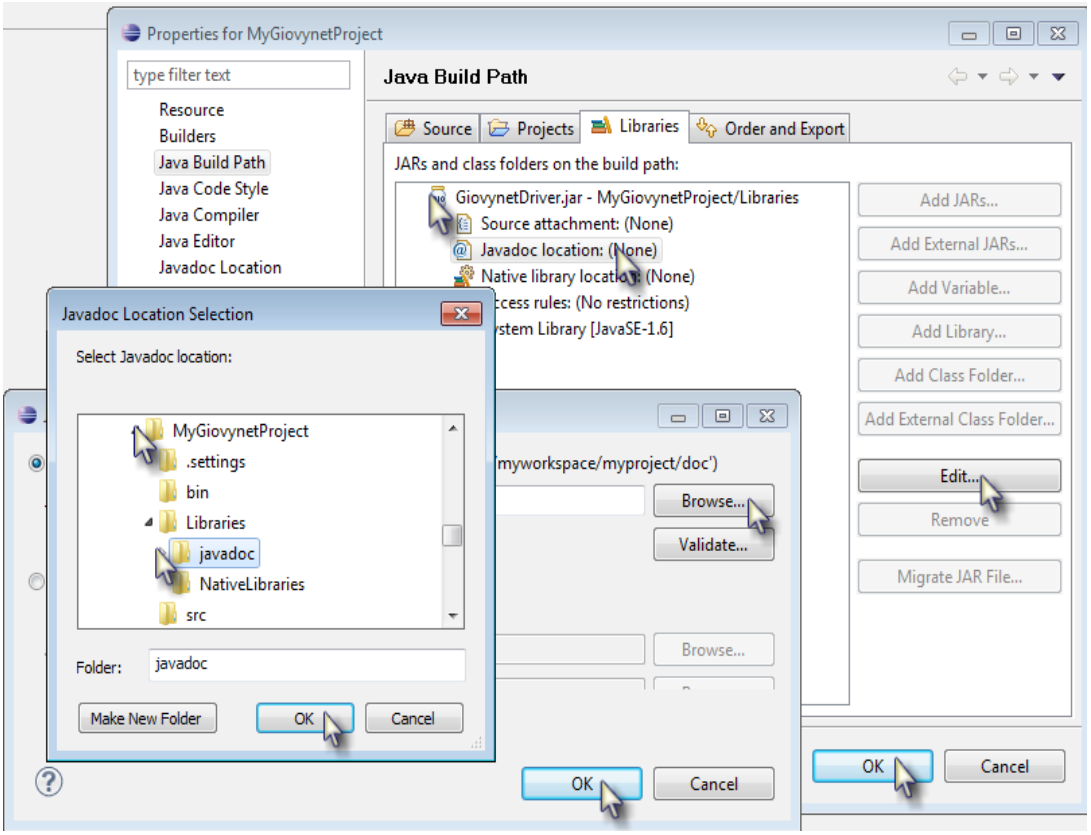


The final step is to copy and paste into the project folder the files: libSerialPort.dll and libSOSerialPort.os, which are within the NativeLibraries folder. The purpose of these libraries is to communicate between Java and the operating system, to see more on this go to the "Architecture" in the first chapter.
After this step you can start building a Java application with Giovynet Driver, the following screen shot shows how the project will be:

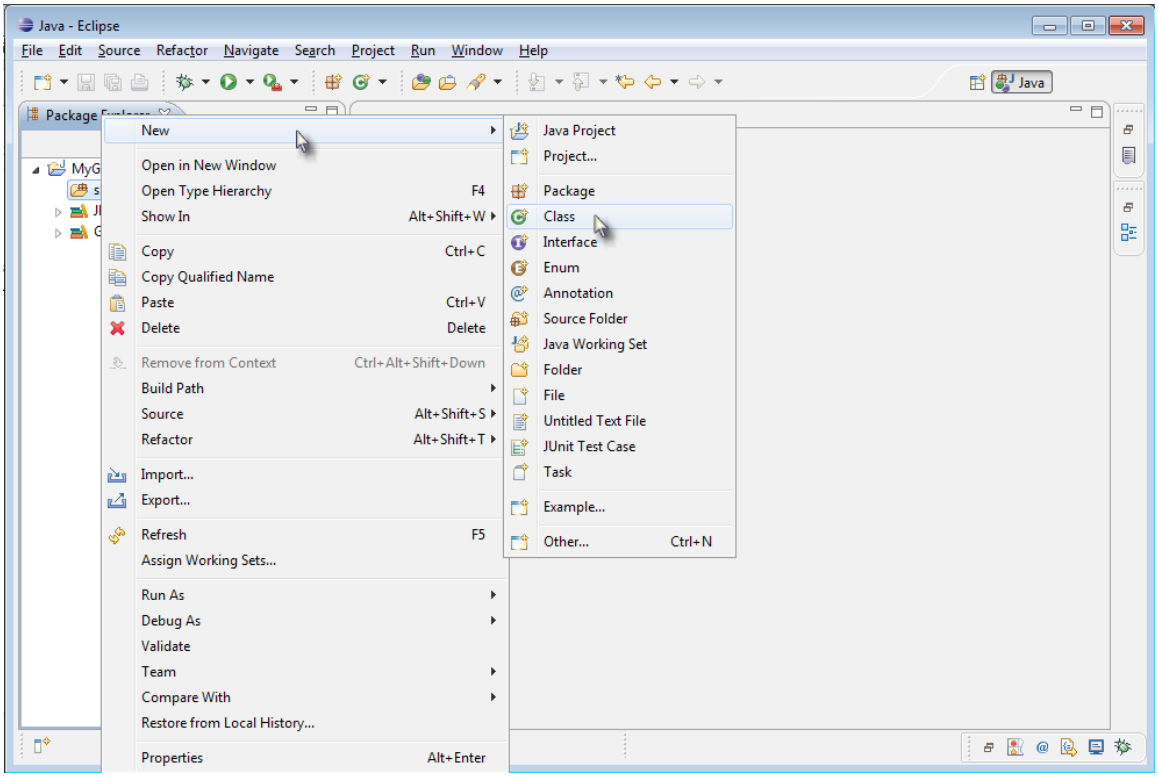
The next step is optional but recommended. It consists of referencing the standard document of Giovynet Driver (information of classes and methods) into Java project. In doing so, Eclipse can display information for classes, methods and parameters of Giovynet Driver when typing source code while pressing the Ctrl and Space. This feature can be helpful for the programmer.

To refer to the documentation of Giovynet Driver in Eclipse, right click the folder of the project and select Build Path | Configure Build Path, then go to Libraries and click on the icon of GiovynetDriver.jar to display the properties tree, then click on Javadoc Location, then go to Edit button and click on it to launch the wizard that allows you to select the Javadoc of Giovynet Driver, as shown in the following image:

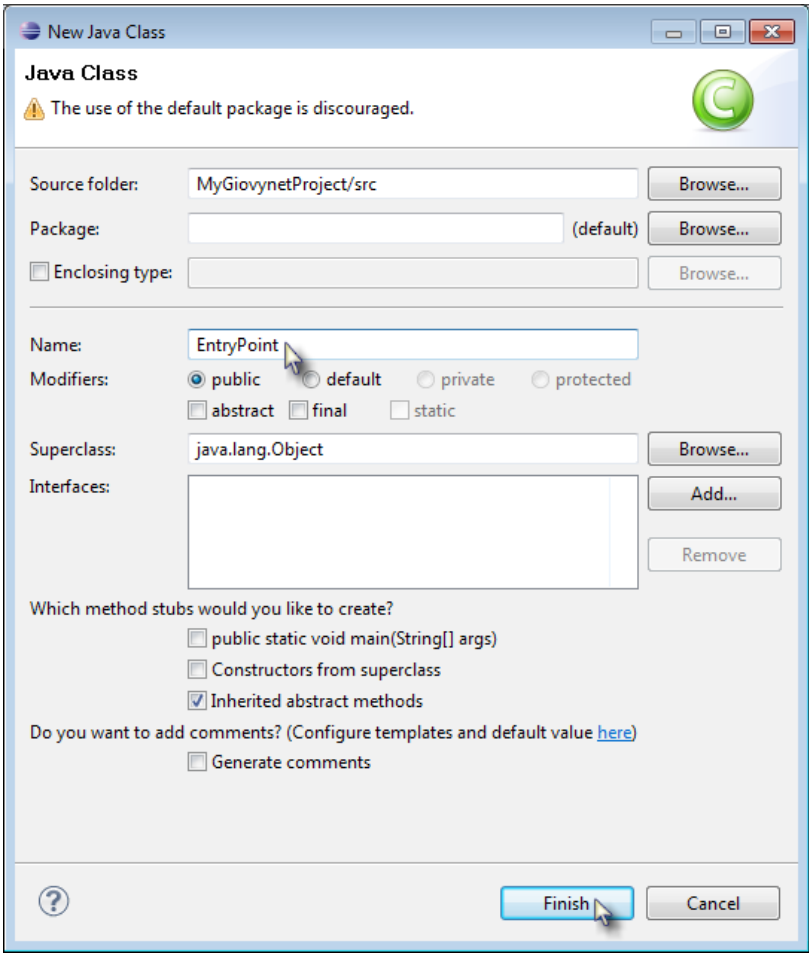




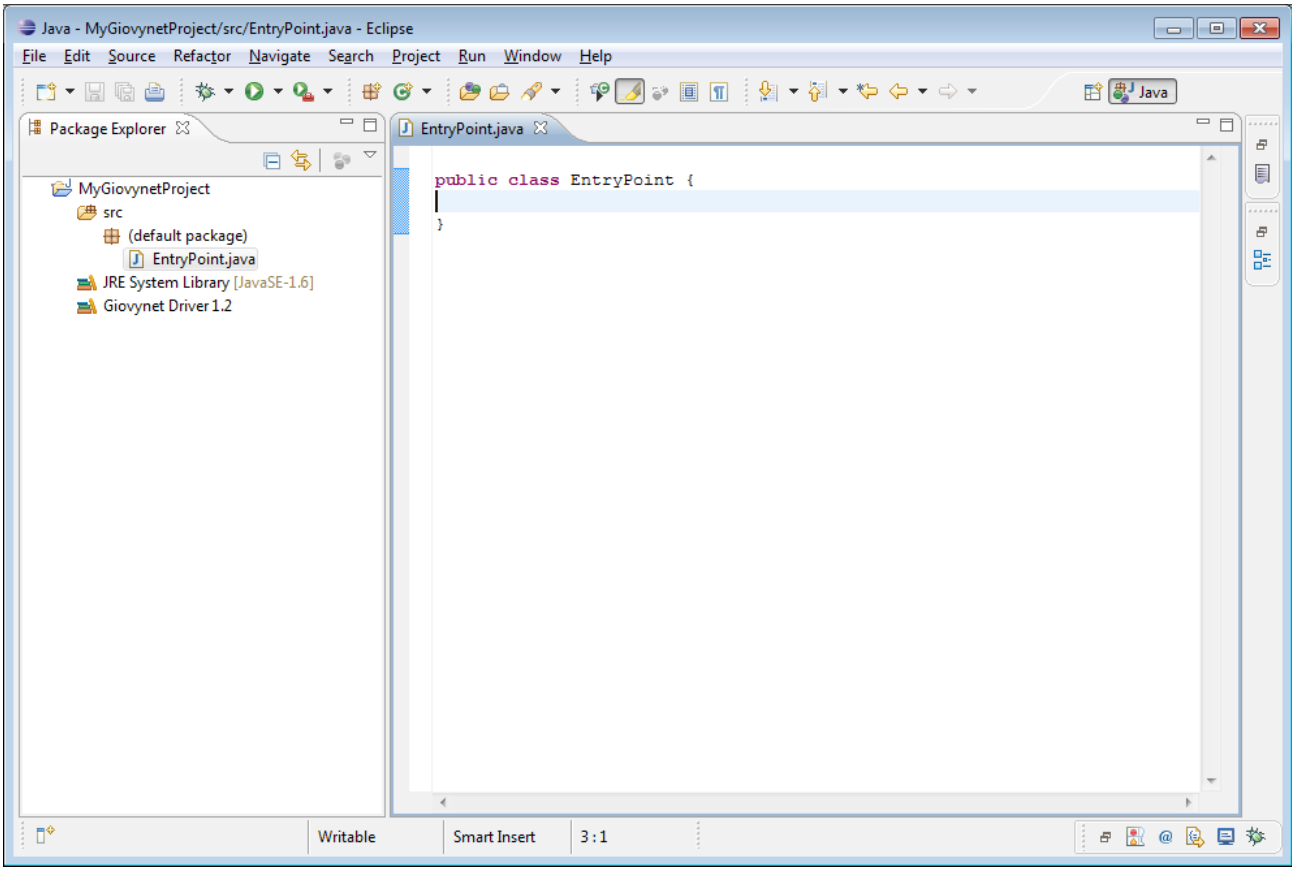
To check the project, you will build a simple application that displays the available serial ports on your PC. Go to the folder of the project, click on it to display the tree of subfolders, then right click on the folder src, and select New | Class, as shown in the image below:



Immediately after clicking on Class their appears a wizard for creating a class. In the Name enter “EntryPoint,” and then click on Finish. Consider the following picture to clarify the idea:



After clicking on the Finish button will create the class, and display the editor to begin typing Java source code, as shown below:

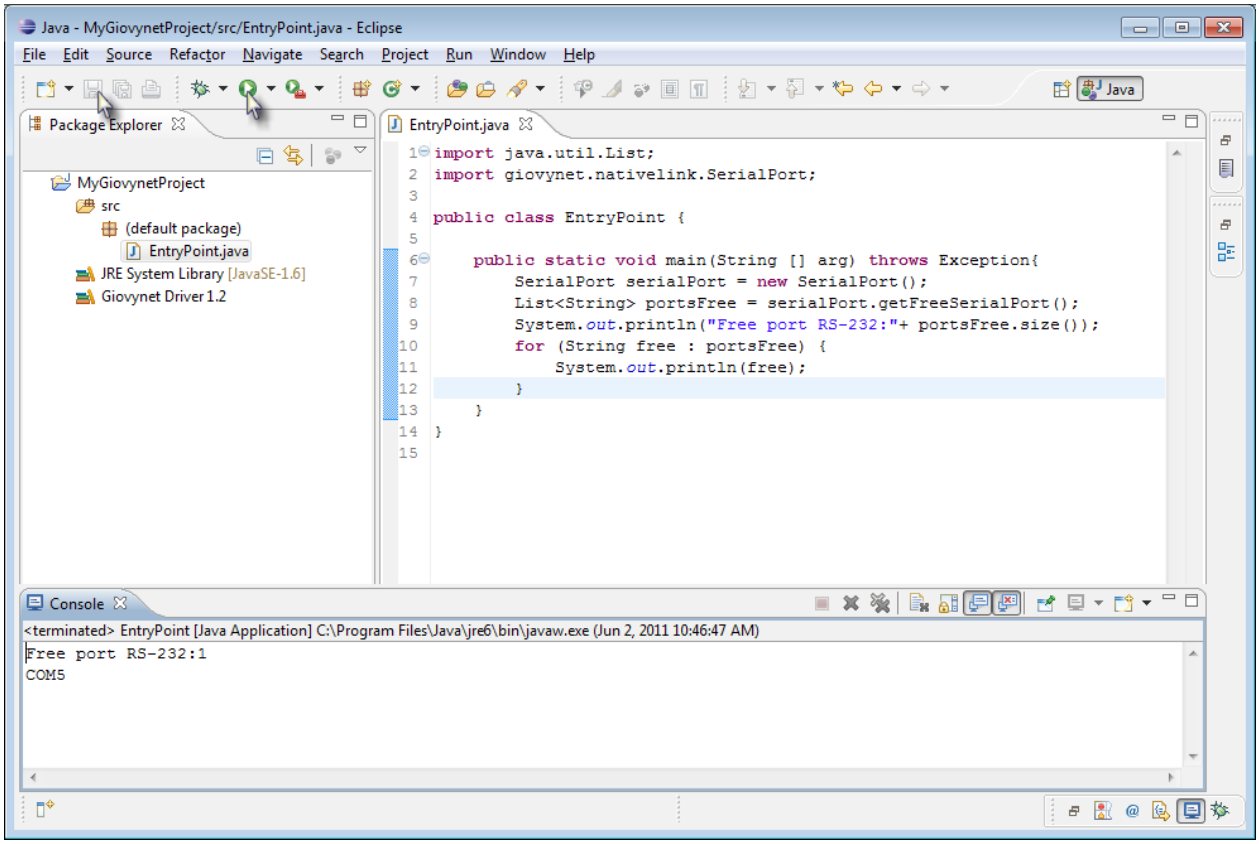


Write into editor the following code:

```
import java.util.List;
import giovynet.nativelink.SerialPort;
public class EntryPoint {
	public static void main(String [] arg) throws Exception{
		SerialPort serialPort = new SerialPort();
		List<String> portsFree = serialPort.getFreeSerialPort();
		System.out.println("Free RS-232 ports:");
		for (String free : portsFree) {
		System.out.println(free);
		}
	}
}
```

After this you must save the class, you could do by pressing Ctrl and S, or press the floppy button on the toolbar.

Finally, run the code by pressing the keys Ctr and F6, or if you prefer, go to the toolbar and press the Run button, (green circle with a white triangle in the middle). Consider the following image to reinforce the text.



The execution of the previous class displays a message (in the console) informing the RS-232 ports available for use; in this case the COM5 is the only free port.

Tip 1:

If running the application and you see the following error:

java.lang.UnsatisfiedLinkError: Can not load library.

This is because the native librararies can not be referenced (**libSerialPort.dll** and **libSOSerialPort.so**), because they are not in the project folder.

To fix this error, copy the files: **libSOSerialPort.so** and **libSerialPort.dll** (located in the folder **NativeLibraries**) and paste these in the folder of the project.

Tip 2:

If running the application and you see the following error:

java.lang.UnsatisfiedLinkError: (Possible cause: architecture word width mismatch).

This is because you can not reference the native libraries (**libSerialPort.dll** and **libSOSerialPort.so**) because these are compiled on a different architecture to the architecture of the PC. To illustrate a possible scenario, assume you want to perform a development for architecture for x64 (64bit), but you are using native files compiled for x86 (32 bit). This will occur because the architectures are not compatible.

The solution is replace current native files by native files of right architecture (i.e., architecture for x64).

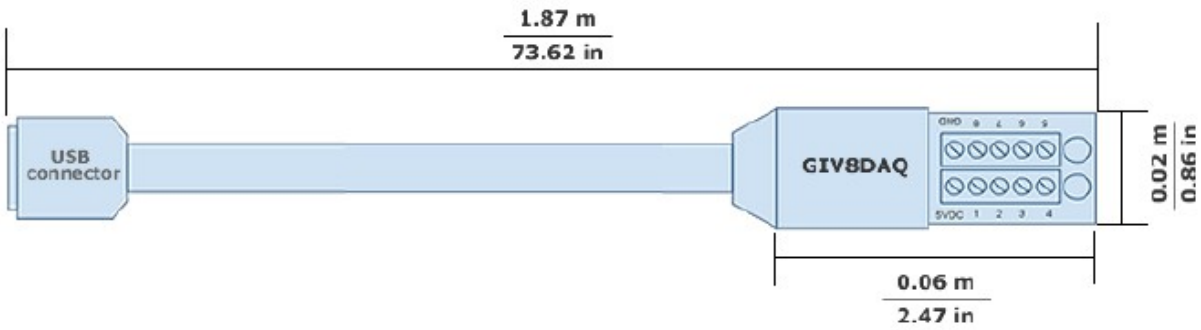
Chapter 4: Port GIV8DAQ

GIV8DAQ port is an interface that connects to the computer via USB. You do not need outside sources to run it because it takes the power of the host PC. For its size and design, easily adapts to external circuits.

Using a GIV8DAQ port, a Java application can do the following:

- Connect external circuits through channels (set programmatically) to function as inputs or outputs in TTL logic (0-5VDC).
- Get analog voltage measurements with a resolution of 10 bits per channel, indicating that for voltages between 0 and 5V it could sense changes in steps of 50mV.
- Obtain temperature measurements per channel by connecting the prior sensor GIV18B20 (explained later).

Dimensions



Features

GIV8DAQ is a set of 8 independent channels, which derives its power from the USB port of the PC. Each channel has the following specifications:

- Digital Output: high state (5VDC) or low state (0VDC), current to 25 mA.
- Digital Input: true (5V), or false (0VDC).
- Analog Input: 10-bit A/D 0-5VDC.
- Temperature: reading using the sensor GIV18B20 (purchased separately), in the range from 67 °F to 257 °F (-55 °C to 125 °C).

Maximum Ratings

Signals applied over the following ranges can cause permanent damage:

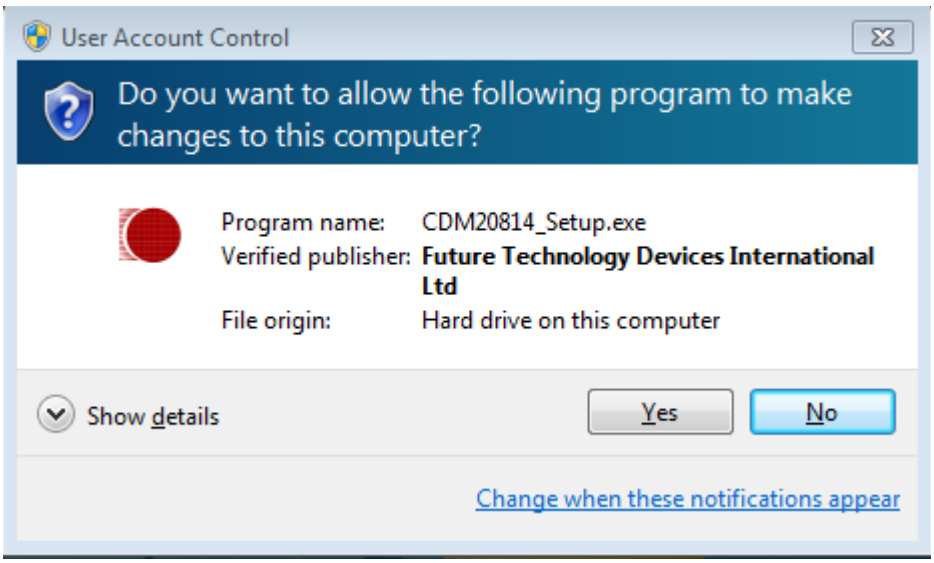
- Operating Temperature: 0°C-70°C
- Voltage on I/O with Respect to Ground: -0.3V to +5.3V
- Current on Any I/O: 25mA

Important Considerations for Windows

It maybe that when connecting the port, Windows will not recognize it. This is because Windows does not have the driver installed for the GIV ports. Therefore, it must be installed manually by double-clicking on the file **CDM208014_Setup.exe** that is included in all distributions of **Giovynet_Driver version 2.0**. This file is a secure and verified applications for Windows, it is distributed by Future Technology Devices International Ltd.

This application installs the drivers onto Windows in order to be able to recognize the GIV ports. This situation does not occur in Linux because the drivers for the ports GIV is included in the kernel so that Linux automatically recognizes these devices.

The following image shows the information window launched by Windows OS after double-clicking on the file **CDM208014_Setup.exe**.



To begin the installation of the drivers click Yes. Then a console will launch that informs you how to install. When finished, the console will close automatically. This action is very fast and to some computers imperceptible. After installation, Windows OS will recognize the port easily.

Programming the Port GIV8DAQ With Eclipse and Java

Before developing Java applications with Giovynet Driver it requires three things:
1) install the JDK, 2) install Eclipse, and 3) know how to make a Java project with Giovynet Driver. If you have not installed these tools, you must go to the first chapter of this book and follow the steps listed there. To learn how to make a Java project with Giovynet Driver, go to Chapter 3 for details.

The goal here is to show you how to easily manipulate the GIV8DAQ from Java. From now on until the end of the chapter the reader will find a series of sections entitled "How to know...?" These sections explain in detail the Java instructions for GIV8DAQ. Also, shown for each

section is a simple snippet of a code that is ready to be run from a class in a Java project with Giovynet Driver.

How do you Know How Many Devices Can Be Instantiated?

As noted in the first chapter, every distribution of Giovynet Driver can manipulate or instantiate a certain number of devices, among which are RS-232 and GIV ports. To find out how many ports could be instantiated for a distribution, use the **getNumDevicesAllowed()** method, it is part of the static class **Info**. This method returns an integer representing the number of devices allowed. The following example shows in the console the number of devices allowed:

```
public static void main (String [] arg) throws Exception {
        System.out.println("Number of devices allowed: "+
Info.getNumDevicesAllowed();
}
```

How to Know if There are Ports Connected?

First you must create an instance of **ScanDevices** class. Then you can use the method **findGIV8DAQ()**, this method returns a list of GIV8DAQ which is connected to the computer. If the list size is zero, it means that no devices are connected. The following code snippet demonstrates the above:

```
public static void main (String [] arg) throws Exception{
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("There is not GIV8DAQ connected");
        }else{
                System.out.println("There are: "+ listDevGIV8DAQ.size()+
                "devices GIV8DAQ connected");
        }
}
```

How to Get an Instance of The Ports Connected?

To perform this task using the method **findGIV8DAQ()** of **ScanDevices** class. This method returns a list of objects of the types GIV8DAQ. Each object represents a device GIV8DAQ; therefore, you can get a reference of a GIV8DAQ port using the method **get(int index)** from the list returned by the method **findGIV8DAQ()**. The following code snippet shows an example of this:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ oneDev = listDevGIV8DAQ.get(0);
                System.out.println("first instance of the list");
        }
}
```

How to Activate a Port?

Each port comes with an activation key, this key must be assigned programmatically after instantiating the port. This is done with the method **activate(String activationkey)** from GIV8DAQ class:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        //Bring GIV8DAQ connected
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                 System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                giv8daq.activate("#####-##");//Port activation
                System.out.println("GIV8DAQ port is active");
        }
}
```

If you try to execute a command without first activating the port, it will launch the following exception:
giovynet.permissions.PermissionsException: The device is inactive. To activate it, first you must use the method: activate (String activation_Key);

How do You Know Which is the Communication Port Associated?

Each GIV port, has an associated single **Com** port of communications, so that you know which **Com** is associated with a GIV8DAQ. You can use the method **getCom()**, from **GIV8DAQ** class-this method returns an object of class Com. The Com object has the method **getPort()** which returns the port name. The following code snippet shows an example of this:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        //Bring the instances of the GIV8DAQ
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                //Port activation
                giv8daq.activate("#####-##");
                //Getting the name of the COM
                String strCom = giv8daq.getCom().getPort();
                System.out.println("The associated port for GIV8DAQ is: "
        +strCom);
        }
}
```

How to Establish a Channel to High (5VDC)?

The GIV8DAQ provides eight channels. Each channel has its own configuration for various uses. To configure a digital output channel to high (5VDC), you use the method **command(Channel ch, DIGITALOUT instruction)**. This method has two parameters. The first parameter is a static attribute called **Channel** from **GIV8DAQ** class. **Channel** has eight static variables, they start with prefix **ch[1-8]** and represent each of the eight channels of the port. The second parameter is a static attribute called **DigitalOut** from **GIV8DAQ** class. **DigitalOut** presents the static variable **set_High**. The following code is used for establishing the channel four high.

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
```

```
        if(listDevGIV8DAQ.size()==0){
                System.out.println("There are no GIV8DAQ connected ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                giv8daq.activate("#####-##");//Activation of the port
        //Command to set on high for channel 4  -->(5VDC)
        giv8daq.command(GIV8DAQ.Channel.ch4,GIV8DAQ.DigitalOutput.set_High);
        System.out.println("Now Channel 4 has 5VDC");
        }
}
```

How to Establish a Channel to Low (0VDC)?

The device GIV8DAQ features eight channels configurable for various purposes. To set a channel output to low (0VDC), use the method command(Channel ch, DigitalOut instruction). This method has two parameters, the first parameter is a static attribute from class GIV8DAQ called Channel. Channel features eight static variables that begin with the prefix ch[1-8], they represent each of the eight channels of the port. The second parameter is a static attribute from GIV8DAQ class called DigitalOutput. DigitalOutput presents the static varible set_Low, which sets the selected channel (first parameter) to low (0VDC). The following snippet of a code is used to establish low (0VDC) the channel four.

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("There are no GIV8DAQ connected");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                giv8daq.activate("#####-##");//Activation of the port
        //Command to set on low for channel 4-->  (0VDC)
        giv8daq.command(GIV8DAQ.Channel.ch4,GIV8DAQ.DigitalOutput.set_Low);
        System.out.println("Now Channel 4 has 0VDC ");
        }
}
```

How to Get the Digital Value (Low or High) From a Channel?

To read the logic state of a channel (high or low) subject it to a potential TTL (0 - 5VDC), you can use the method command (Channel ch, DigitalInput instruction). This method has two

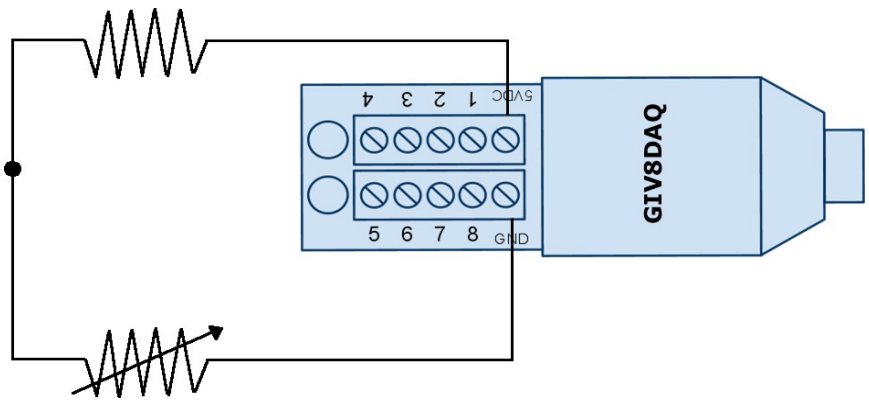
parameters. The first parameter is a static attribute from GIV8DAQ class called Channel. Channel which features eight static variables that start with prefix ch[1-8], they represent each of the eight channels of the port. The second parameter is a static attribute of Class GIV8DAQ called DigitalInput. DigitalInput presents the static variable get_Digital_Input. This variable indicates that it will retrieve the logical value from the channel; remember that the first parameter indicates the channel. The following snippet of code is used to obtain the logic state from channel 1.

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                //Activation of the port
                giv8daq.activate("######-##");
        //Command to get the logic state of channel 1
        Boolean statusCH1 =
        giv8daq.command(GIV8DAQ.Channel.ch1,GIV8DAQ.DigitalInput.get_Digital_Input);
        System.out.println("The logical status of channel 1 is:  "+statusCH1);
        }
}
```

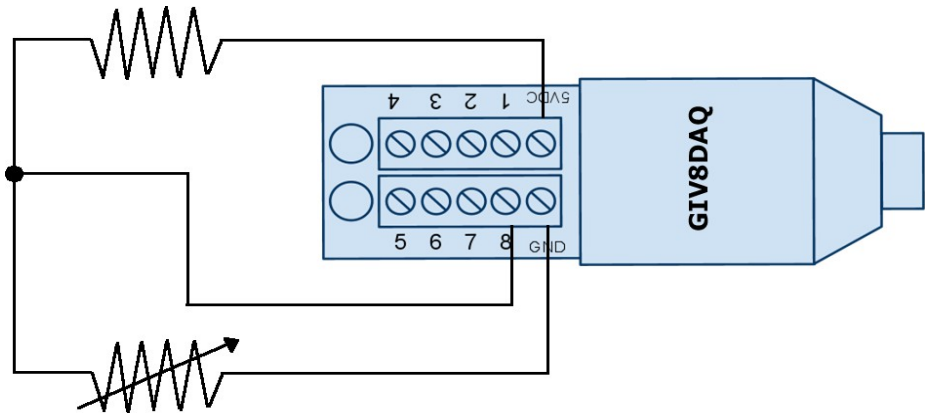
If the terminals of the channel 1 are subjected to a potential 5VDC, the state will be true. if the terminals of the channel 1 are subjected to a potential 0VDC the state will be false.

How to Get a Reading Voltage From a Channel?

Each channel from GIV8DAQ can read voltages. To illustrate this with an example, suppose you have a series circuit composed of a fixed resistor and a variable resistor, one end of the circuit is connected to the GND terminal and the other end is connected to the 5V terminal, as illustrated in the following image.



Assume that is required to read the voltage of the variable resistor. To do this connect the junction between resistors to the channel of GIV8DAQ, then programmatically you can get the value of the voltage in the channel. The following figure illustrates this example using the channel 8.



To read voltage (in volts) in a channel, use the method **command(Channel ch, Analog instruction)**, which returns a value in format String. This method has two parameters, the first parameter is a static attribute from **GIV8DAQ** class, called **Channel**. **Channel** features eight static variables that start with prefix **ch[1-8]**, they represent each of the eight channels of the port. The second parameter is another attribute of Class **GIV8DAQ** called **Analog**. **Analog** provides static variable **get_Voltage_Measurement**. This variable indicates that it will retrieves voltage value from channel (first parameter). The following piece of a code is used to obtain the voltage present on channel 8.

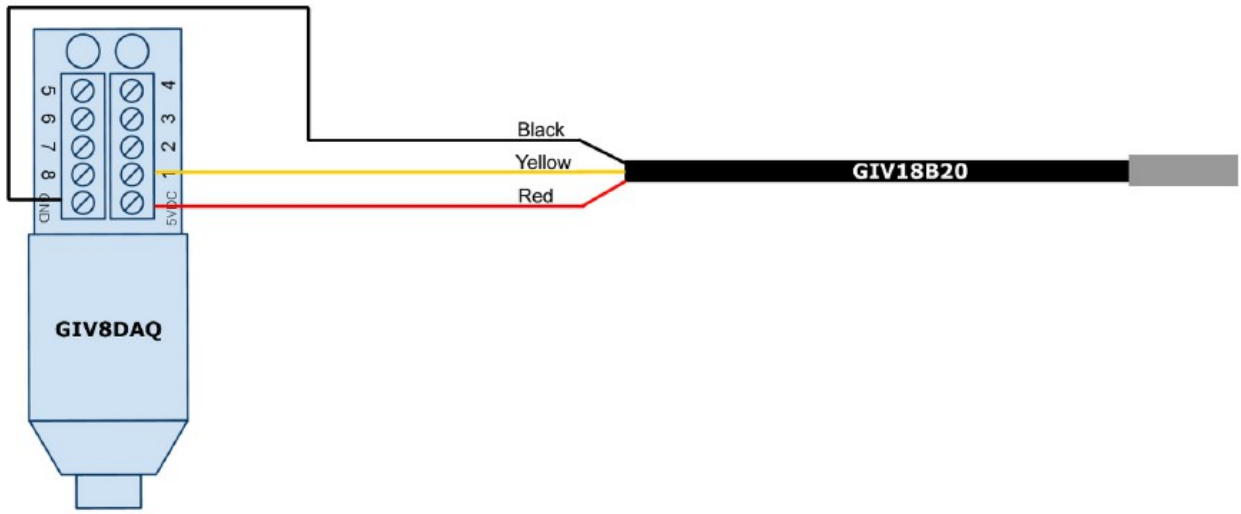
```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                //Activation of the port
                giv8daq.activate("#####-##");
        //Command that reads the voltage on channel 8
        String voltageCH8 =
        giv8daq.command(GIV8DAQ.Channel.ch8,GIV8DAQ.Analog.get_Voltage_Measurement);
        System.out.println("The voltage in the channel 8 is: "+ voltageCH8);
        }
}
```

How to Connect the Temperature Sensor GIV18B20 to a Channel?

For each channel of GIV8DAQ you could connect a temperature sensor GIV18B20, to do this follow these steps:

1. Connect the red cable to 5V terminal.
2. Connect the black cable to GND terminal.
3. Finally, connect the yellow cable to the desired channel.

To reinforce the idea consider the following image, it illustrates the connection of a sensor GIV18B20 to the channel 1.



The GIV18B20 sensor has a water resistant probe, it is made of stainless steel, it has a length of 118 inches (3 meters), and it is able to provide measurements in the range from 67 to 257 °F (-55 to 125 ° C). For more technical information about this module, contact Giovynet.com.

How to Obtain a Temperature Reading?

To get a reading from the sensor temperature GIV18B20, connected to a channel, use the method **command(Channel ch, Analog instruction)** that returns the value in the format of a String. This method has two parameters, the first parameter is a static attribute from **GIV8DAQ** class called **Channel**. **Channel** features eight static variables that start with prefix **ch[1-8]**, they represent each of the eight channels of the device. The second parameter is a

static attribute from GIV8DAQ class called Analog. Analog has the static variable get_Temperature_Measurement, this variable indicates the temperature value obtained for the selected channel in the first parameter.

Keep in mind that the first data read will be 999.99 °F (°C), subsequent readings will return valid data.

The following code snippet shows four temperature readings that returns a GIV18B20 sensor connected to the channel 3.

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                //Activation of the port
                giv8daq.activate("#####-##");
                String valueTemp ="";
                //Command to get the temperature reading of a sensor GIV12B20 -
                //connected to Channel 3.
                //The first reading is discarded

giv8daq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
                for (int i = 0; i < 4; i++){
                        Thread.sleep(1000);//Waiting time of 1 second
        valueTemp =

giv8daq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
                        System.out.println(i+"Temperature Date: "+ valueTemp);
                }
        System.out.println("Temperature readings on channel one is: "+ valueTemp);
        System.out.println("Reading time set for response: "+        giv8daq.getTimeInquiryCH3());
        }
}
```

How to Set the Mode Temperature Reading in Fahrenheit or Celsius?

The default format read for all channels is Fahrenheit, however it is possible to change the format of reading to Celsius. To set the read mode, you use the method **command(TemperaturaReadingMode instruction)**, which provides the **TemperaturaReadingMode** parameter. This parameter is a static attribute from **GIV8DAQ** class, it has the static variables: **set_Fahrenheit** and **set_Celsius**, they are used to set the mode of reading in Celsius or Fahrenheit. For example the following code snippet shows how to set the mode of reading in Celsius:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                //Activation of the port
                giv8daq.activate("######-##");
                GIV8DAQ giv8dq = listDevGIV8DAQ.get(0);
                //Sets the format of reading in Celsius for all channels.
                giv8dq.command(GIV8DAQ.TemperaturaReadingMode.set_Celsius);
                String valueTemp ="";
                //Command to get the temperature reading of a sensor GIV12B20 -
                //connected to channel 3.
                valueTemp =
        giv8dq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
        System.out.println("Temperature reading on channel one is: "+ valueTemp);
        }
}
```

Important Considerations in Temperature Reading

When you execute a command for reading temperature or voltage, internally **Giovynet Driver** sends a query to the port **GIV8DAQ**, which receives and processes the query and stores the result in a memory area. After a time of 100 milliseconds (set by default), **Giovynet Driver** reads the memory area and returns the answer to Java. It is unusual but could happen that the processing time and the reception time exceed the reading timeout set. This is especially true for temperature readings, because it adds the processing time of the GIV8DAQ with the

processing time of the sensor GIV18B20. If this happens the data obtained is wrong, it is out of range: 999.99°F/C. It is recommended that you discard the other values and stay in the ranges of 67 to 257 º F (-55 to 125 ° C), because these are the ranges that are possible for GIV18B20. On the other hand, it is also possible programmatically to extend processing and reception times for each channel using the method:**setTimeInquiryCH[1-8](int timeInMilliseconds)** from GIV8DAQ class.

For example, the following is a snippet of a code to establish 500 milliseconds for reading time in channel 3.

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ.");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                //Activacion del puerto
                giv8daq.activate("#####-##");
                //Command to set the format for reading in Celsius -
                //for all channels.
                giv8daq.command(GIV8DAQ.TemperaturaReadingMode.set_Celsius);
                //Command to set 500 ms for reading response on channel 3
                giv8daq.setTimeInquiryCH3(500);
                String valueTemp ="";
                //Command to get the temperature reading of a sensor GIV12B20,
                //connected to Channel 3.
valueTemp = giv8daq.command(GIV8DAQ.Channel.ch3,GIV8DAQ.Analog.get_Temperature_Measurement);
        System.out.println("The temperature reading on channel three is: "+ valueTemp);
        System.out.println("Reading time set for response: "+
        giv8daq.getTimeInquiryCH3());
        }
}
```

How to Know the Number of Instantiated Ports?

Because not all distributions of Giovynet Driver can instantiate the same number of devices[2], it may be necessary find out programmatically the current number of instances from GIV8DAQ. You can to do this by using the static method **getNumberOfInstancesGIV8DAQ()** from **Info** class.

```
public static void main (String [] arg) throws Exception {
```

2 Refer to Chapter 1, Distributions and Licensing.

```
        System.out.println("Number of GIV8DAQ instantiated currently: "+
        Info.getNumberOfInstancesGIV8DAQ());
}
```

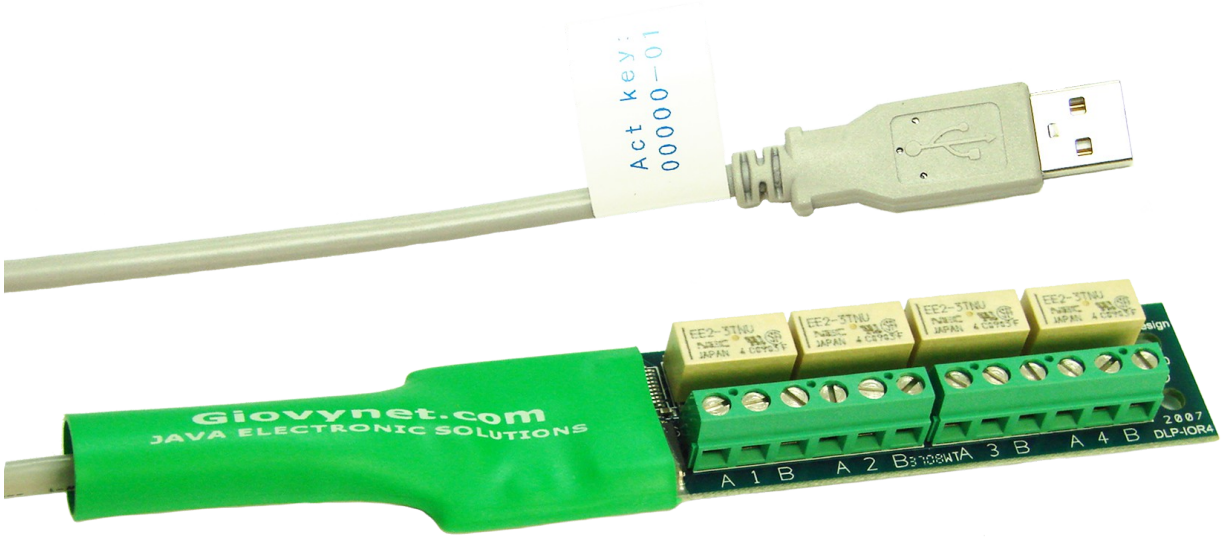
How do You End an Instance of a Port?

To end an instance of GIV8DAQ, use the close() method from GIV8DAQ class. The following example ends the first instance, returned by the method findGIV8DAQ().

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV8DAQ> listDevGIV8DAQ =sd.findGIV8DAQ();
        if(listDevGIV8DAQ.size()==0){
                System.out.println("No connected GIV8DAQ");
        }else{
                GIV8DAQ giv8daq = listDevGIV8DAQ.get(0);
                System.out.println("Number of GIV8DAQ objects: "+
                Info.getNumberOfInstancesGIV8DAQ());
                //Ends the instance giv8daq
                giv8daq.close();
                System.out.println("Number of GIV8DAQ objects: (after closing): "+
        Info.getNumberOfInstancesGIV8DAQ());
        }
}
```

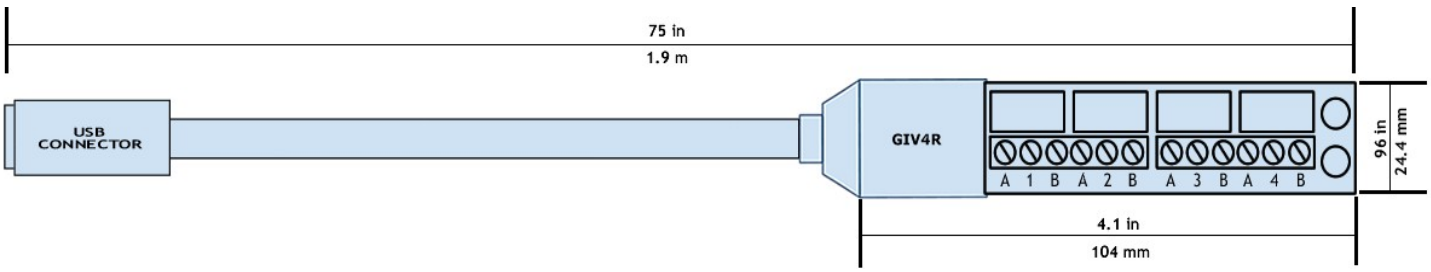
Chapter 5: Port GIV4R

The port GIV4R is an interface that is connected to the computer via USB. You do not need outside sources to operate, because it takes the power of the host PC, for its size and design easily adapts to external circuits. It has four relays, each relay has three terminals. The terminal of the center represents the common terminal of the relay and is marked with a number from 1 to 4. Each common terminal, can be connected programmatically to the left terminal marked with "A" or to the right terminal marked with "B." The cable length is 73 inches (1.8 m).



Each relay is independent in its operations and maintains its state in case of disconnection or shutdowns of the PC. The GIV4R port is designed for users who want to enable or disable circuits of more power than the PC itself.

Dimensions



Specifications

GIV4R USB port is a device with four independent relays in operation that derives its power from the PC. Each relay has the following specifications:

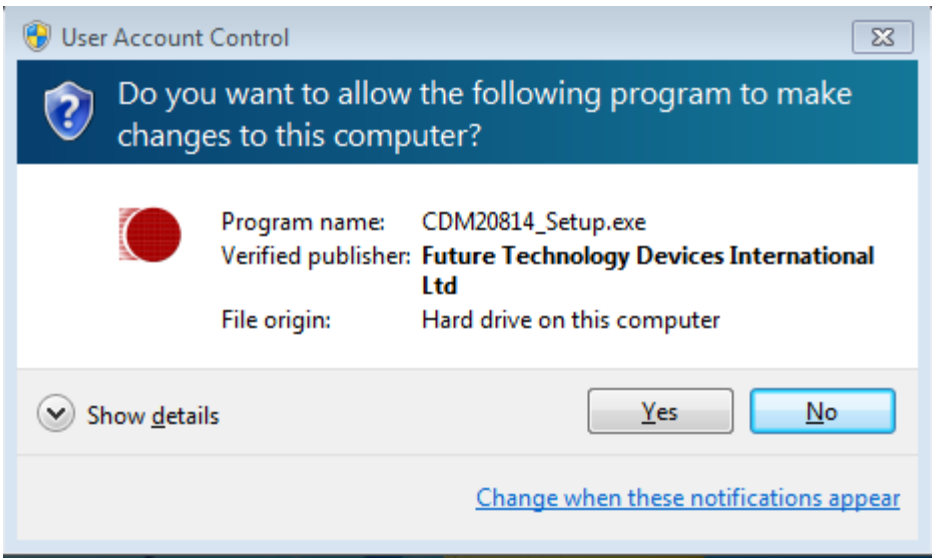
- Contact Ratings: 60W, 125VA
- Max Switching Voltage: 220VDC, 250 VAC
- Max Switching Current: 2A
- Max Carrying Current: 2A
- Operating Temperature: 0-70°C

Exposing the port to values higher or out of range from the values listed above may permanently damage it.

Important Considerations for Windows

It may be that when connecting the port, Windows will not recognize it. This is because Windows does not have installed the driver for the GIV ports. Therefore, it must be installed manually by double-clicking on the file **CDM208014_Setup.exe** which is included in all distributions of Giovynet Driver version 2.0. This file is a secure and verified application for Windows, and is distributed by: Future Technology Devices International Ltd. This application installs on Windows the drivers to it recognized GIV ports. This situation does not occur in Linux because the driver for the GIV ports is included in the Linux kernel, so that it recognizes these devices automatically.

The following image shows the information windows launched by Windows OS after double-clicking on the file CDM208014_Setup.exe:



To begin the installation click Yes, then a console will launch reporting the installation process. The end is closed automatically, this action is very fast and some computers imperceptible. After installing it, Windows OS recognizes the port smoothly.

Programming the Port GIV4R with Eclipse and Java

Before developing Java applications and Giovynet Driver, three things are required: 1) Install the JDK, 2) Install Eclipse and 3) Know how to create a Java project with Giovynet Driver. If you have not installed these tools, go to the first chapter of this book and follow the steps listed there. If you want to know how to form a Java project with Giovynet Driver, refer to Chapter 3.

The goal here is to show you how to easily manipulate the GIV4R port from Java. From now on and until the end of this chapter the reader will find a series of sections entitled "How to know...?" These sections explain in detail each of the instructions for GIV4R, there are also simple code snippets for each section ready to be executed from a class in a Java project Giovynet Driver.

How to Know How Many Devices can be Instantiated?

As noted in the first chapter, every distribution of Giovynet Driver allows you to manipulate or instantiate a certain number of devices between RS-232 and GIV ports. To find out how many ports could be instantiated for a distribution, use the **getNumDevicesAllowed()**, it belongs to the static class **Info**. This method returns an integer representing the number of devices allowed. The following example shows in console the number of devices allowed:

```
public static void main (String [] arg) throws Exception {
        System.out.println("Number of devices allowed: "+ Info.getNumDevicesAllowed());
}
```

How to know if There are Ports Connected?

You must first create an instance of **ScanDevices** class, then you can use the method **findGIV4R()**, this method returns a list of objects GIV4R connected to the computer. If the list size is zero, it means that no devices are connected. The following code snippet demonstrates the above:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R = sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                System.out.println("No connected GIV4R");
        }else{
                System.out.println("There are: "+ listDevGIV4R.size()+
                " GIV4R connected");
        }
}
```

How to Obtain an Instance of the Devices Connected?

To perform this task using the method **findGIV4R()** of **ScanDevices** class. This method returns a list of instances or objects of type GIV4R, each object represents a device GIV4R, therefore you can get a reference of a GIV4R port, using the method **get(int index)** from the list returned by the method **findGIV4R()**. The following code snippet shows an example of this:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R =sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                System.out.println("No connected GIV4R");
        }else{
                GIV4R oneDev = listDevGIV4R.get(0);
                System.out.println("first instance of the list");
        }
}
```

How to Activate a Port?

Each port comes with an activation key, this key must be assigned programmatically after instantiating the port. This is done with the command **activate(String activationkey)** from GIV4R class:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R =sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                System.out.println("No connected GIV4R");
        }else{
                GIV4R giv4r = listDevGIV4R.get(0);
                //Port activation
                giv4r.activate("#####-##");
                System.out.println("Was activated first port GIV4R.");
        }
}
```

If you try to execute a command without activating the port, it will launch the following exception: giovynet.permissions.PermissionsException: The device is inactive. To activate it, first you must use the method: activate (String activation_Key);

How do you Know Which is the Communication Port Associated With?

Each GIV port, has an associated single **Com** port communications, to know which is the **Com** associated with a GIV8DAQ, you can use the method **getCom()**, from **GIV4R** class, this method returns an object of class **Com**. The **Com** object has the method **getPort()** which returns the port name. The following code snippet shows an example of this:

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R =sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                System.out.println("No connected GIV4R");
        }else{
                GIV4R giv4r = listDevGIV4R.get(0);
                //Activación del puerto
                giv4r.activate("#####-##");
                System.out.println("Was activated first port GIV4R.");
                //Getting the name of the COM
                String strCom = giv4r.getCom().getPort();
                System.out.println("The port associated with GIV4R is: "
                +strCom);
        }
}
```

How to Connect the Comun to Terminal “A” in a Relay?

Each relay has three terminals. The terminal of the center represents the common connection, it can be connected programmatically to terminal A (left) or to the terminal B (right).

To connect the common terminal of a relay to terminal A (left), use the method **command (Relay relay, Connec.to_A)** from **GIV4R** class. The first parameter represents the relay and the common terminal, the second parameter is where it will connect (terminal A or terminal B). The following code snippet shows how to connect the common terminal of the relay 1 with the terminal A.

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R =sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                System.out.println("No GIV4R connected ");
        }else{
                GIV4R giv4r = listDevGIV4R.get(0);
                //Port activation
                giv4r.activate("#####-##");
                System.out.println("Was activated first port GIV4R.");
                giv4r.command(GIV4R.Relay._1,GIV4R.Connect.to_A);
                System.out.println("Relay 1 was connected to terminal A.");
        }
}
```

How to Connect the Comun to Terminal “B” in a Relay?

To connect the common terminal of a relay to terminal B (left), use the method **command(Relay relay, Connec.to_B)** from **GIV4R** class. The first parameter represents the relay and the common terminal, the second parameter is where it will connect (terminal A or terminal B). The following code snippet shows how to connect the common terminal of terminal "B" on the relay "4".

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R =sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                System.out.println("No connected GIV4R");
        }else{
                GIV4R giv4r = listDevGIV4R.get(0);
```

```
                //Port activation
                giv4r.activate("#####-##");
                System.out.println("Was activated first port GIV4R.");
                giv4r.command(GIV4R.Relay._4,GIV4R.Connect.to_B);
                System.out.println("Relay 4 was connected to terminal B.");
        }
}
```

How to Know the Number of Ports Instantiated?

Because not all distributions of Giovynet Driver it can instaciate the same number of devices, may be necessary to get the current number of GIV8DAQ instances. Using the method **getNumberOfInstancesGIV8DAQ()** of the static class **Info**, you can do that.

```
        public static void main (String [] arg) throws Exception {
                System.out.println("Total GIV8DAQ instantiated currently : "+
        Info.getNumberOfInstancesGIV8DAQ());
        }
```

How do you End an Instance of a Port?

Para finalizar una instancia de GIV4R, use el método **close()** de la clase GIV4R. En el siguiente ejemplo se finaliza la primera instancia, retornada por el método findGIV4R().

```
public static void main (String [] arg) throws Exception {
        ScanDevices sd = new ScanDevices();
        List<GIV4R> listDevGIV4R =sd.findGIV4R();
        if(listDevGIV4R.size()==0){
                 System.out.println("No connected GIV4R");
        }else{
                GIV4R giv4r = listDevGIV4R.get(0);
                System.out.println("GIV4R instanciados: "+
                Info.getNumberOfInstancesGIV4R());
                //Finaliza la instancia giv4r
                giv4r.close();
                System.out.println("Number of GIV8DAQ objects: (after closing): "+
                Info.getNumberOfInstancesGIV4R());
        }
}
```

Chapter 6: Sending and Receiving ASCII Characters via RS-232

This chapter consists of a series of sections entitled "How to know?" These sections explain in detail each of the instructions for a RS-232 port. There are also simple code snippets for each section ready to be executed from a class in a Java project with Giovynet Driver.

Determining Which Serial Ports are Free

First you must create an object of type **giovynet.nativelink.SerialPort**, then the **getFreeSerialPort()** method is used to get a **List<String>** of free ports:

```
public static void main(String[] args)throws Exception{
        SerialPort serialPort = new SerialPort();
        List<String> portsFree = serialPort.getFreeSerialPort();
        for (String free : portsFree) {
                System.out.println(free);
        }
}
```

How to Configure the Serial Port?

First you create an object of type **giovynet.serial.Parameter**, which features the following default settings:

- Port = COM1
- Baud Rate = 9600
- Byte Size = 8
- Stop Bits = 1
- Parity = N (none)

To work with the above configuration, you need to instantiate an object of type **giovynet.serial.Com**, with the **parameter** object in the constructor. Thus, opens the serial port COM1 and you are ready to work.

```
Parameters parameter = new Parameters();
Com com = new Com(parameter);
```

To change the settings for default, use the set methods of the object parameter before instantiating the class **giovynet.serial.Com**. For example, to set the COM2 port at a rate of 460800 bps, you can use the following instructions:

```
Parameters param = new Parameters();
param.setPort("COM2");
param.setBaudRate(Baud._460800);
Com com = new Com(param);
```

How to Send Data?

To make this work, Giovynet Driver uses **giovynet.serial.Com** class which has two methods presented below:

1. sendArrayChar(char[] data) : void

This method is used to send elements of an "array of char." The time interval that is sent to each item is determined by the method-setMinDelayWrite(int milliseconds) from **giovynet.serial.Parameters** class, by default time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code is used to send elements in an array, every 100 milliseconds:

```
public static void main(String[] args)throws Exception{
        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(100);
        Com com1 = new Com(param);
        char[] data = {'1','A','2','B'};
        com1.sendArrayChar(data);
        com1.close();
}
```

Running this code, after 100 milliseconds will send the char '1 ', then another 100 milliseconds it will send the char 'A', then another 100 milliseconds it will send the char '2' and so on until the last char stored in the array data.

2. sendSingleData(overloaded) : void

This method is used to send three items of type: char, String or Hex. The item will be sent after the time specified in the method **setMinDelayWrite (int milliseconds)** from **giovynet.serial.Parameters** class. Default time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code shows how to send the next data: **'a'**, **41**, **"S"** and **0xff** in intervals of 100 milliseconds:

```
public static void main(String[] args)throws Exception{
        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(100);
        Com com1 = new Com(param);
        com1.sendSingleData('a');
        com1.sendSingleData(41);
        com1.sendSingleData("S");
        com1.sendSingleData(0xff);
        com1.close();
}
```

Running the above code will send you after 50 milliseconds, the item **'a'** followed by another 50 milliseconds, then it will send you the item **41**, then another 50 milliseconds it will send you the item **"S"** and finally, it will send you the item **0xff**.

How to Get Data?

To perform this task Giovynet Driver uses the **giovynet.serial.Com** class, which has four methods presented below:

1. receiveSingleChar() : char

This method is used to receive ASCII data, then the time prescribed by the method **setMinDelayWrite(int milliseconds)** from **giovynet.serial.Parameters** class. Default time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. It is not recommended for receiving control characters, such as <ACK>, <NACK>, <LF>, etc. For example, the following code is used to receive 10 char elements, every 50 milliseconds, and print it to the console:

```
public static void main(String[] args){

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);

        char data;
        for(int x=0; x<10; x++){
                data=com1.receiveSingleChar();
                System.out.println(data);
        }
```

```
        com1.close();
}
```

2. receiveSingleCharAsInteger() : int

This method is used to receive data in ASCII integer representation, after the time set by the method **setMinDelayWrite(int milliseconds)** from **giovynet.serial.Parameters** class. Default time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. This method is recommended to receive control characters such as <ACK>, <NACK>, <LF>, ... etc. For example, the following code receives 10 data type Integers, every 50 milliseconds, and prints it to the console:

```
public static void main(String[] args)throws Exception{
        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);
        int data;
        for(int x=0; x<10; x++){
                data=com1.receiveSingleCharAsInteger();
                System.out.println(data);
        }
        com1.close();
}
```

3. receiveToString(int amount) : String

This method is used to receive multiple items and return them in an ASCII String. The receipt of each item is made after the time set by the method **setMinDelayWrite(int milliseconds)** from **giovynet.serial.Parameters** class. Default time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code is used to receive a String of 10 items, each item is received every 50 milliseconds.

```
public static void main(String[] args)throws Exception{
        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);

        String data;
        data=com1.receiveToString(10);
        System.out.println(data);
        com1.close();
}
```

4. receiveToStringBuilder(int untilAmount, StringBuilder stringBuilder) : void

This method is used to receive various elements of Strings and the data you receive is stored in StringBuilder object. The number of items you receive at the StringBuilder are passed as its first parameter. The receipt of each item is made after the time set by the method **setMinDelayWrite(int milliseconds)** from **giovynet.serial.Parameters class.** Default time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code gets 20 data (String) every 30 milliseconds and stored in an StringBuilder object.

```
public static void main(String[] args)throws Exception{
        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(30);
        Com com1 = new Com(param);

        StringBuilder stringBuilder = new StringBuilder;
        com1.receiveToStringBuilder(20,stringBuilder);
        com1.close();
}
```

How to Send Control Characters?

To send control characters, it is recommended to use the decimal or hexadecimal representation using the **sendSingleData()** method from **giovynet.serial.Com** class. For example, the following code sends the ACK symbol (acknowledge):

```
public static void main (String [] args) throws Exception {
        Parameters param = new Parameters ();
        param.setPort ("COM1");
        param. setBaudRate (Baud._9600)
        Com1 = new Com Com (param);

        int ack = 6 //Integer representation of ACK in ASCII code is 6
        data = com1.sendSingleData (ack);
        com1.close ();
}
```

How to Get Control Characters?

To receive control characters, you are recommended to obtain them in decimal or hexadecimal representations, using the method **receiveSingleCharAsInteger()**, from **giovynet.serial.Com** class. For example, the following code prints out "OK" when it receives the ACK symbol.

```
public static void main(String[] args)throws Exception{
        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        Com com1 = new Com(param);
        int ack=06;//Integer representation of ACK in ASCII code is 06
        int data=0;
        while(true){
                data=com1.receiveSingleCharAsInteger();
                if (data==ack){
                        System.out.println("OK");
                        break;
                }
        }
        com1.close();
}
```

How to Implement Thread to Receive Data Independently?

Suppose you want to build an application each time you read or receive certain character (or characters) for a serial port, performing a task such as displaying a message on a screen, sending email, performing a database query, or sending a message via the serial port. This could be done using a thread.
In Java there are two ways to implement a thread. One way is to inherit the **Thread** class and the other way is to implement the **Runnable** interface. Both implements the **run()** method with the tasks you want to run independent of the primary task or main thread. As an example, the code below shows two classes that work separately but are part of the same application. The EntryPoint class launches the second class (ReadiangThread) and sends data through the serial port "COMUSB3;" and the ReadiangThread class gets or reads data from the serial port and so prints them on the console.

```
import giovynet.serial.Com;
import giovynet.serial.Parameters;
public class EntryPoint {

        private static Com com;
```

```
	public static void main(String[] args) throws Exception {
		System.out.println("APPLICATION START");
		Parameters param = new Parameters();
		param.setPort("COMUSB3"); //Select a port
		com = new Com(param);//COM Instance
		Thread threadReading  = new Thread(new ThreadReading (com));
		//Start thread
		threadReading .start();

		//Sends 10 times every 0.5 seconds the integer 2
		for (int x = 0; x < 10; x++) {
			com.sendSingleData('A');
			System.out.println("Sends 'A' by the COMUSB3 port.");
			Thread.sleep(270);
		}

		//Stop thread
		threadReading .interrupt();
		//Assigns null  to be a candidate for the garbage collector
		threadReading = null;
		//Calls the garbage collector to free up memory.
		System.gc();
		System.out.println("APPLICATION END");

	}

}



import giovynet.serial.Com;
public class ThreadReading  implements Runnable {

	private Com com;

	public ThreadReading (Com com) {
		this.com = com;
	}

	@Override
	//This method is executed when is call ThreadReading.start()
	public void run() {
		System.out.println("  -->Starts thread .");
		try{
			char data = 0;
			while (true) {//Make an infinite loop
				data= com.receiveSingleChar();//reads the port
				if (data !=0){
```

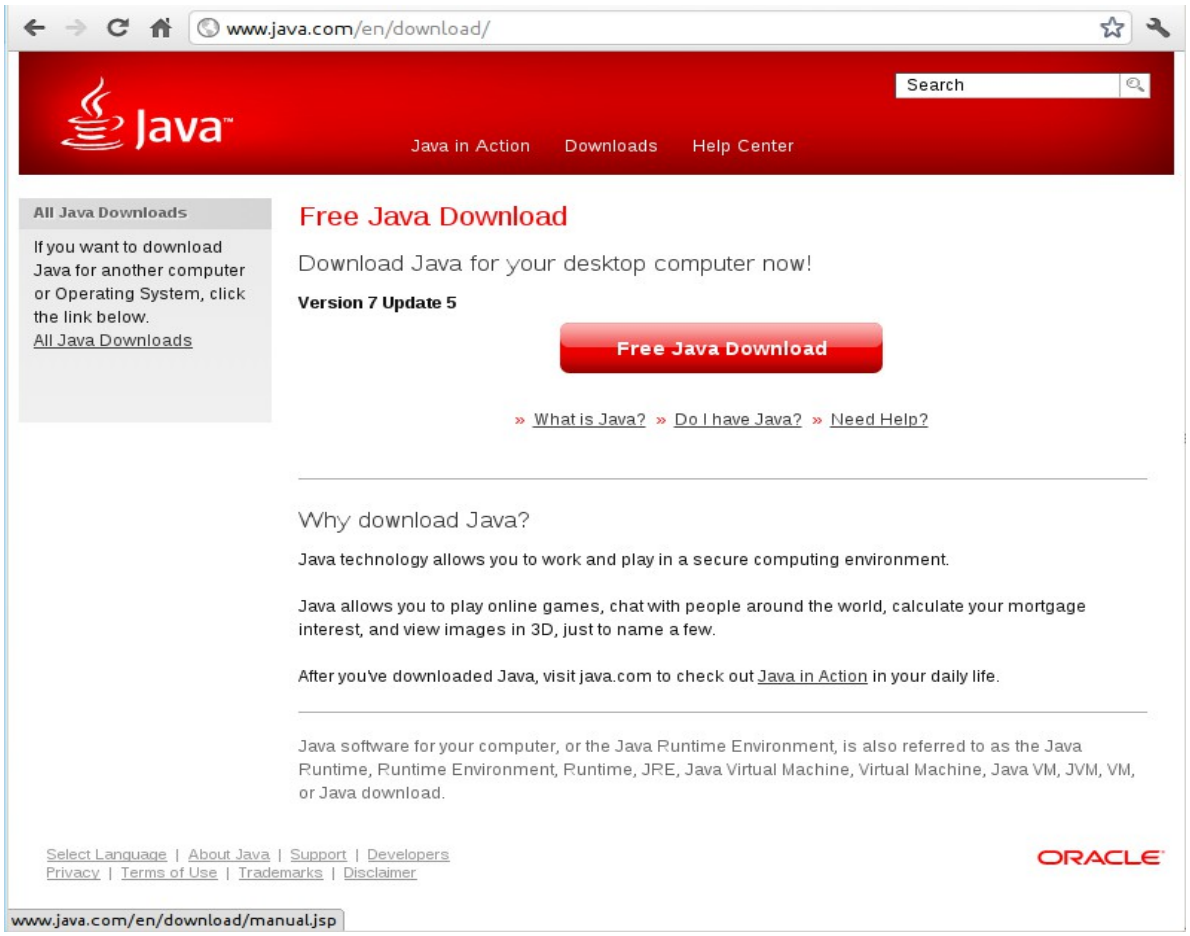
```
                                        do {
                                        System.out.println(" Reading from thread: "+
                                        data);
                                                Thread.sleep(250);
                                                data = com.receiveSingleChar();
                                        }while (data != 0);
                                }
                        }
                }catch (Exception e) {
                        System.out.println("   <--Ends thread");
                }
        }

}
```

Chapter 7: Application Distributions

Eclipse IDE easily creates an executable file, but it is not the only file required to run Java applications with Giovynet Driver. Along with this file you must add the native files of GIovynet Driver and prior to application execution it must be installed JRE (Java Runtime Environment). This chapter describes how to perform each of these steps.

What is and How to Install the JRE?

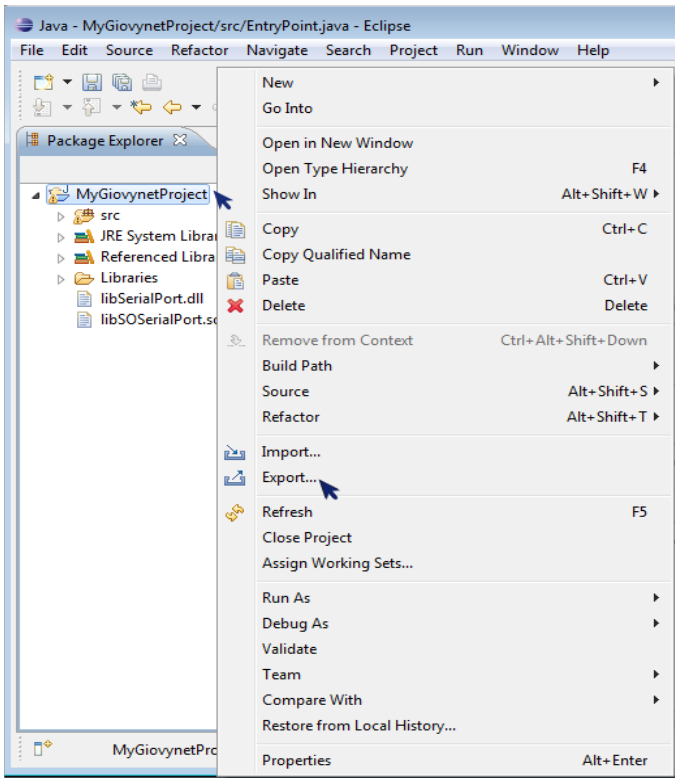
Java application with Giovynet Driver could be run on Linux and Windows. For this to be possible, both Linux and Windows should have installed an application known as JRE. In operating systems like Ubuntu and Debian JRE is already installed by default, but this does not happen in Windows. For this operating system, you must install the JRE manually. To do this simply go to the website: http://www.java.com/en/download/ and download and then run Java application. The following figure shows the website that lets you download the application:

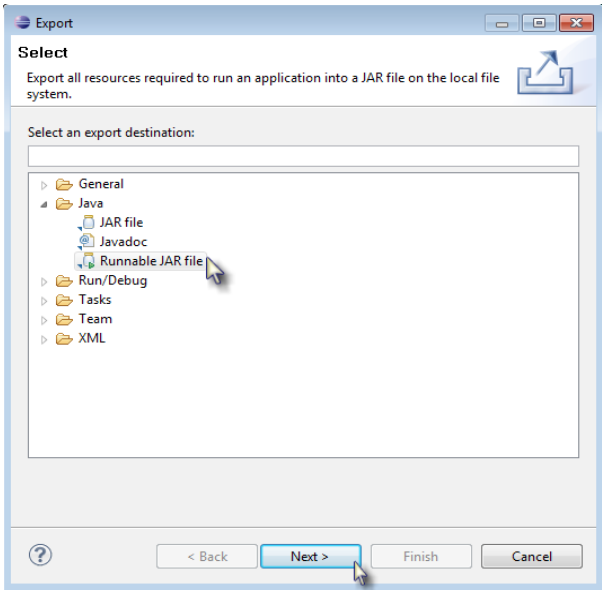


How to Create an Executable JAR with Eclipse?

To create an executable Java, the application must have at least one class with the **public static void main (String [] arg)** method, this forms the entry point of the application. This means that the execution of your Java file is equivalent to run the **main** method of a preselected class. This class can instantiate the other classes that make up the main components of the program.

To create an executable Java in Eclipse IDE, right click on the folder of the project developed, then select export. Window export options appears. Select the folder Runnable JAR File from Java and click on Next as shown below:

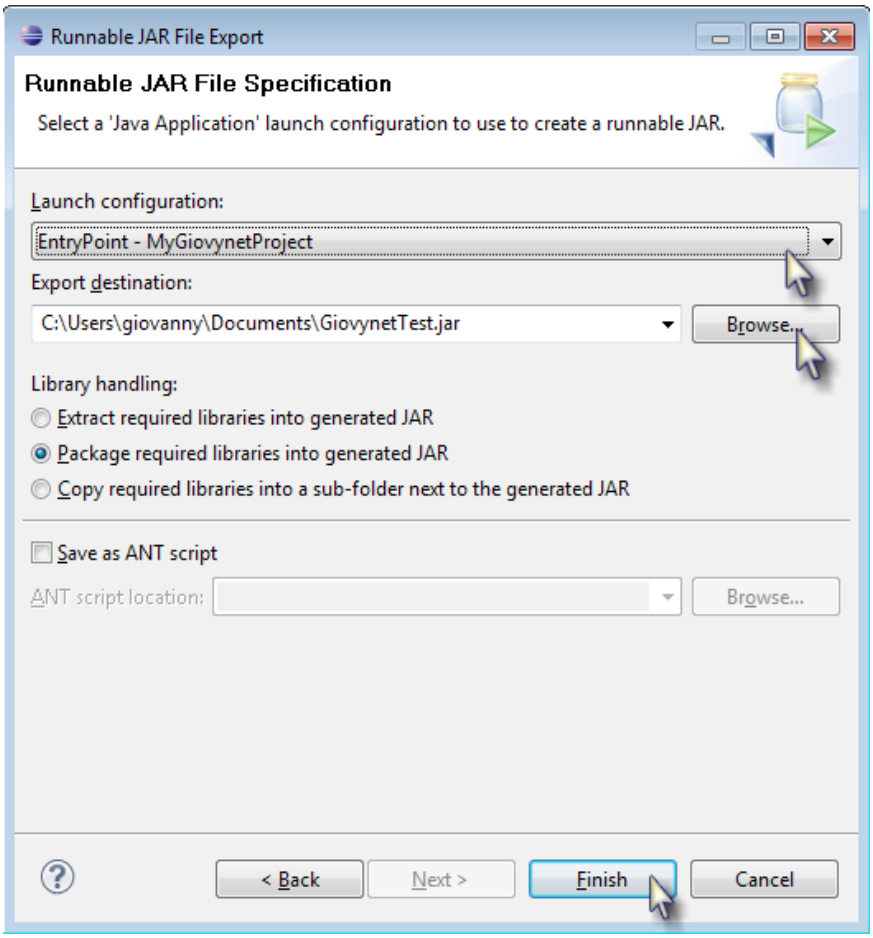




Then you will see a window with several export options. The option Launch configuration appears; select the Class with a **main** method to be called when running the Java file. Select Export destination and type the name of the Java file. Option **Library Handling** must be selected as **Package required libraries generated JAR Into**, which allows classes and methods from Giovynet Driver to be used in the construction of the application, and are packed into the executable Java file. As a final option, "Save as ANT script," this option is optional and allows you to create an

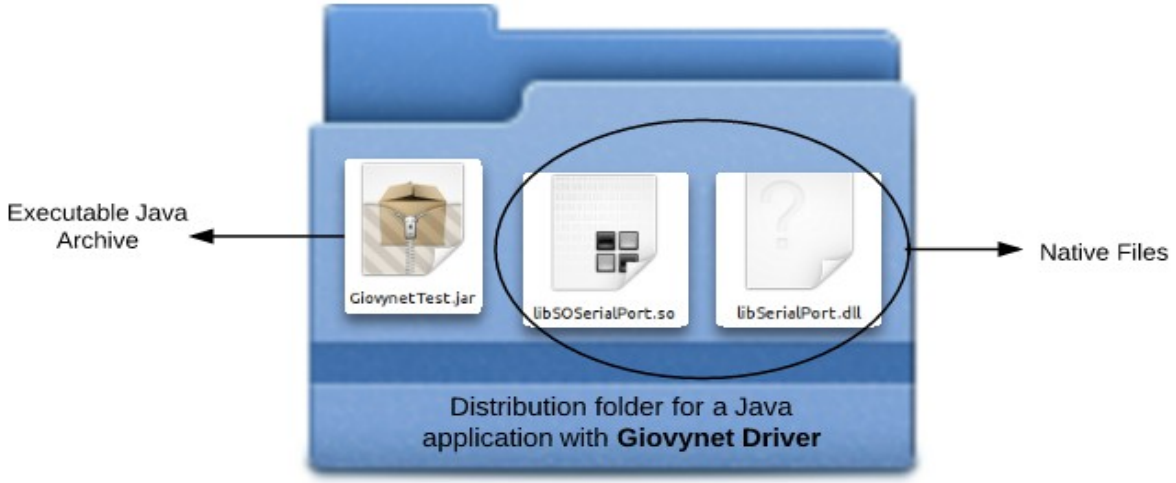
xml file, which is used by the software tool for automating software build processes ANT.

After selecting the export options and clicking on the button Finish, it creates the Java file executable. The figure below shows the options selected for the creation of a Java executable file named GiovynetTest.jar, whose main method is in class EntryPoint.



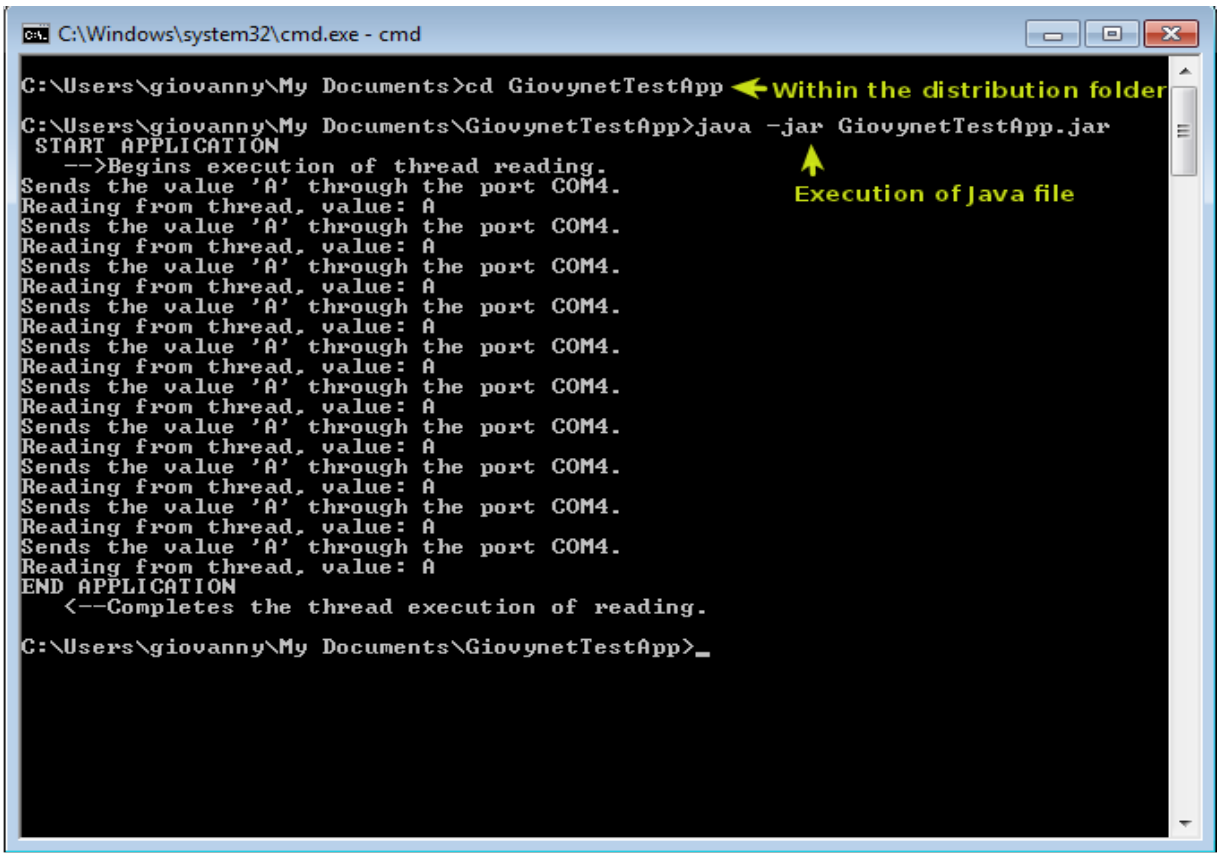
How to Create a Distribution Folder?

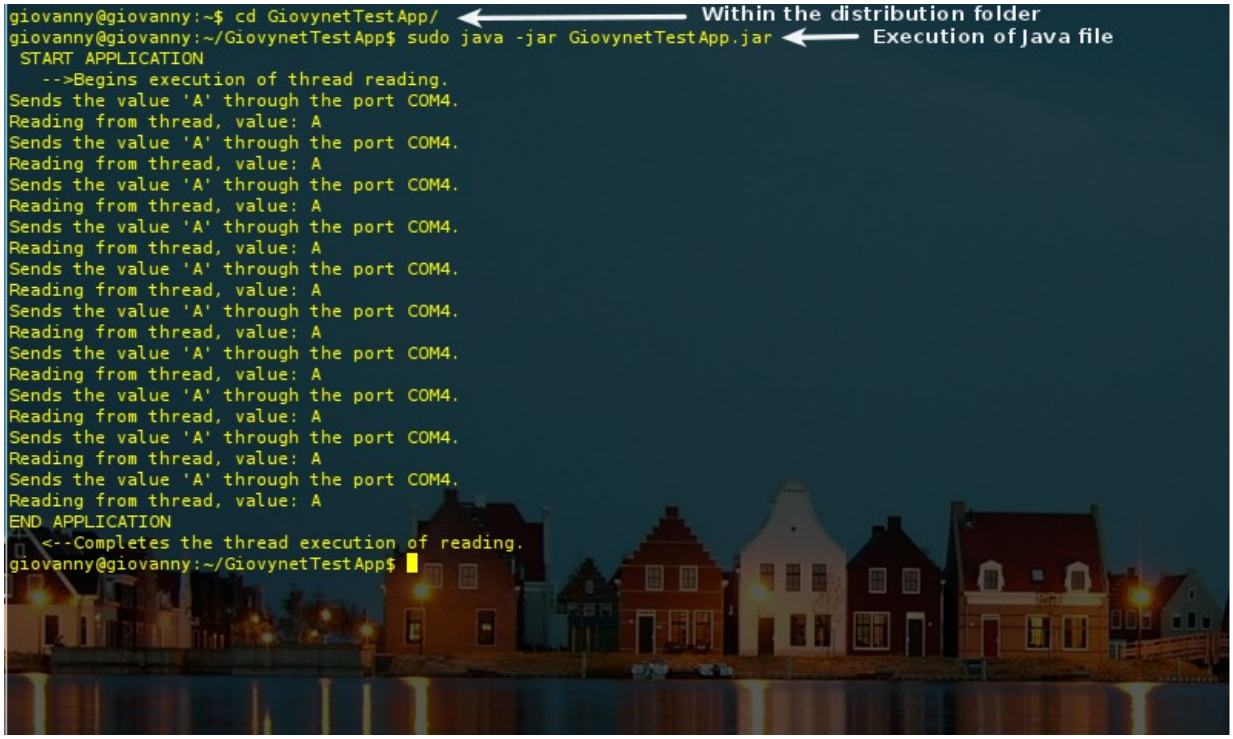
The Java executable file alone can not run the application, because this file will attempt to bind the native files: libSerialPort.dll and libSOSerialPort.so. If these files are not present then launch the exception: **UnsatisfiedLinkError: Can not load library**. So the right way to distribute an application is to create a folder that has the java executable with native files. This way when you run the java file, this can reference these files because they are in the same location. The graph below illustrates this idea.



How to Start an Application From the Distribution Folder?

To launch a Java application with Giovynet Driver while you are in Windows, go to the distribution folder and run the java file by double-clicking on it. Regardless if you are in Windows or Linux you are still running from the console by the command **java-jar** followed by the name of the Java executable file. To run applications that don't have graphical interface is recommended using **java-jar** command. The images below shows this idea:




giovanny@giovanny:~$ cd GiovynetTestApp/
Within the distribution folder
giovanny@giovanny:~/GiovynetTestApp$ sudo java -jar GiovynetTestApp.jar
Execution of Java file
START APPLICATION
-->Begins execution of thread reading.
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
Sends the value 'A' through the port COM4.
Reading from thread, value: A
END APPLICATION
<--Completes the thread execution of reading.
giovanny@giovanny:~/GiovynetTestApp$

Chapter 8: Frequent Exceptions

Here are a few exceptions that can be thrown by a Java application with Giovynet Driver for execution and compilation times. It also describes the reasons for which they are raised, and how to fix them.

java.lang.UnsatisfiedLinkError: Can't load library.

This occurs because the native files can not be referenced (libSerialPort.dll and libSOSerialPort.so), because these are not found in the folder of the project. To fix this error, copy the files **libSOSerialPort.so** and **libSerialPort.dll**, located in the folder NativeLibraries and paste them in the project folder.

java.lang.UnsatisfiedLinkError: (Possible cause: architecture word width mismatch).

This occurs because you can not reference the correct native files (**libSerialPort.dll and libSOSerialPort.so**) because these are compiled on different architecture from the architecture of the PC. For instance, you want to perform an architecture development for x64 (64-bit), but instead you use native files compiled for x86 (32-bit). As the architectures are not compatible when you try to compile, this error will occur. The solution is replace native files for the right native files from the architecture of the computer.

giovynet.permissions.PermissionsException: You have exceeded the number allowed of devices instantiated for this license.

This exception occurs because you try to instantiate a number of devices greater than those permitted for your distribution of Giovynet Driver. As an example, we have the following scenario: Suppose you have a Giovynet Driver For Personal Use (it is allowed to use one device) , and you want to use it, for instance, at the same time with the Serial and GIV8DAQ ports (two devices). Then you try to run this application, it will launch this exception because the number of instances (devices) is greater than the permitted by Giovynet Driver For Personal Use. The solution is to programmatically control the application to not exceed the number of instances allowed, or obtain a distribution that supports more devices.

giovynet.devices.DeviceException: The device was closed, you should re-instantiate this object.

This exception occurs because you tried to use a device that was previously closed. The solution is to re-instanciate the device, after you close it, if you want to use it again.
giovynet.permissions.PermissionsException: The device is inactive.

To activate it, first you must use the method: activate(String activation_Key). This exception occurs when trying to use a device without activating it. The solution is to activate the GIV port before you use it. You should use the method: **activate(String activation_Key)**. The **activation_key** parameter corresponds to the activation key, which is attached close to the USB connector in the port.

S.D.G.