Appendix

Serial Communication
with Java

This appendix dedicated to explaining how to do serial communication with java using an external library, with the same examples from the original chapter on the serial communication.

Content

Prepared by Student: Ahmed Moustafa Amin Aboulrous

Supervised by Dr. Mokhtar Ahmed

01. Why java doesn`t support Serial communication by default?

Because of Java is platform-independence, serial interfacing is difficult. Serial interfacing requires a standardized API with platform-specific implementations, which is difficult for Java.

Unfortunately, Sun didn't pay much attention to serial communication in Java. Sun has defined a serial communication API, called JavaComm, but an implementation of the API was not part of the Java standard edition. Sun provided a reference implementation for a few, but not all Java platforms. Particularly, at the end of 2005 Sun silently withdrew JavaComm support for Windows. Third party implementations for some of the omitted platforms are available. JavaComm hasn't seen much in the way of maintenance activities, only the bare minimum maintenance is performed by Sun, except that Sun apparently responded to pressure from buyers of their own Sun Ray thin clients and adapted JavaComm to this platform while dropping Windows support.

This situation, and the fact that Sun originally did not provide a JavaComm implementation for Linux (starting in 2006) led to the development of the free-software RxTx library. RxTx is available for a number of platforms, not only Linux. It can be used in conjunction with JavaComm (RxTx providing the hardware-specific drivers), or it can be used stand-alone.

RxTx supports more platforms than the existing JavaComm implementations. Recently, RxTx has been adopted to provide the same interface as JavaComm, only that the package names don't match Sun's package names.

02. Available Libraries for serial communication in java

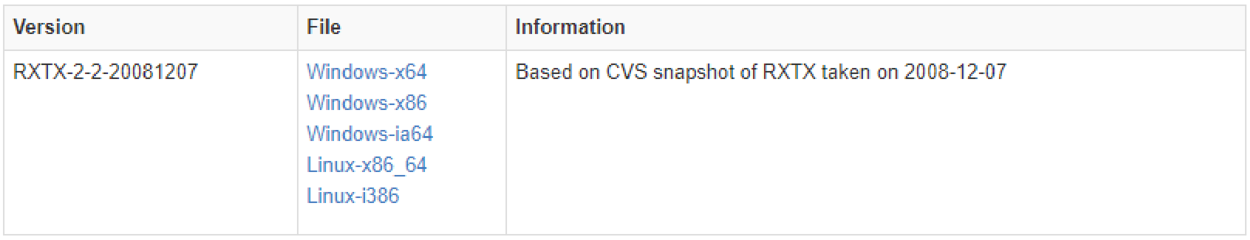
	Pricing	**Notes**
JavaComm	Free	Deprecated
RXTX	Free	Replacement to JavaComm
JSSC	Free	Simple Serial Connector
JSerialComm	Free	alternative to RxTx
serialio	Paid	Commercial Java Serial Port API

Reasons for choosing RXTX {Free, Simple, Easy to use, Follows the original java library}

03. How to setup RXTX library in windows (a onetime setup)

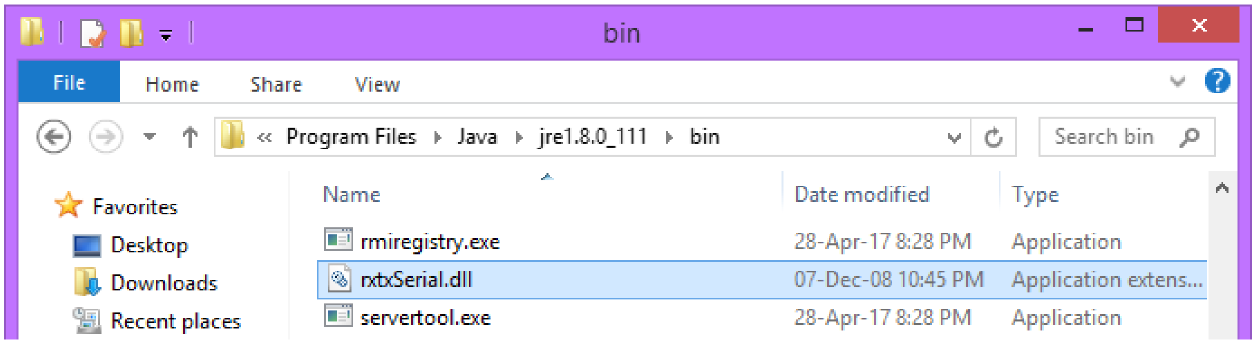
1- Go to the URL [http://fizzed.com/oss/rxtx-for-java]

2- Under Downloads choose the file suitable for your machine



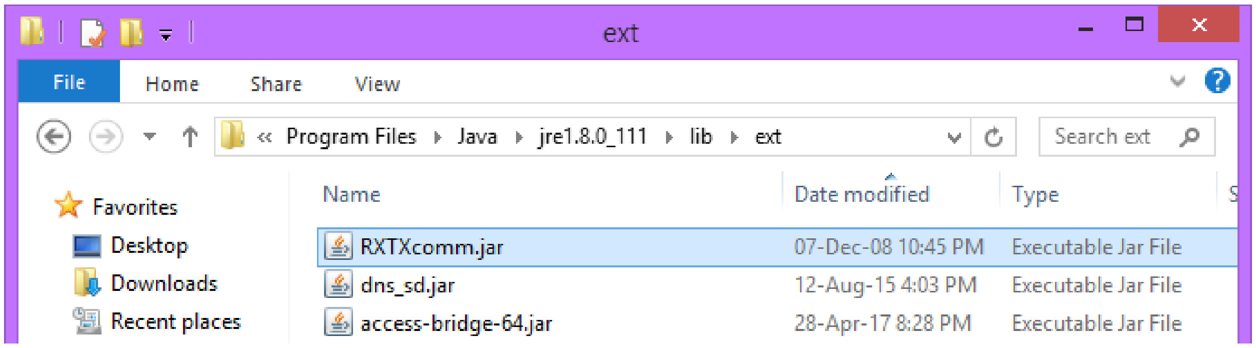
Version	File	Information
RXTX-2-2-20081207	Windows-x64 Windows-x86 Windows-ia64 Linux-x86_64 Linux-i386	Based on CVS snapshot of RXTX taken on 2008-12-07

3- Copy **rxtxSerial.dll** into >> **C:\Program Files\Java\<jre-version>\bin**



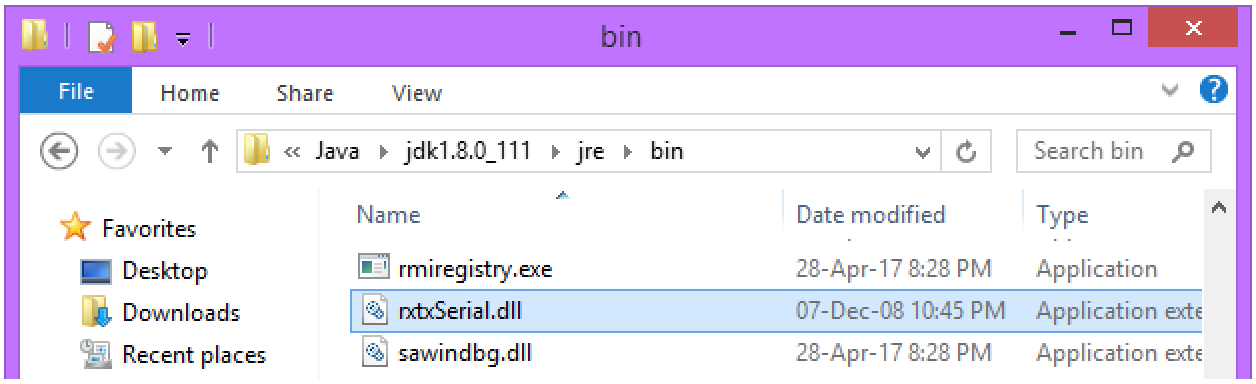
Name	Date modified	Type
rmiregistry.exe	28-Apr-17 8:28 PM	Application
rxtxSerial.dll	07-Dec-08 10:45 PM	Application extens...
servertool.exe	28-Apr-17 8:28 PM	Application

4- Copy **RXTXcomm.jar** into >> **C:\Program Files\Java\<jre-version>\lib\ext**



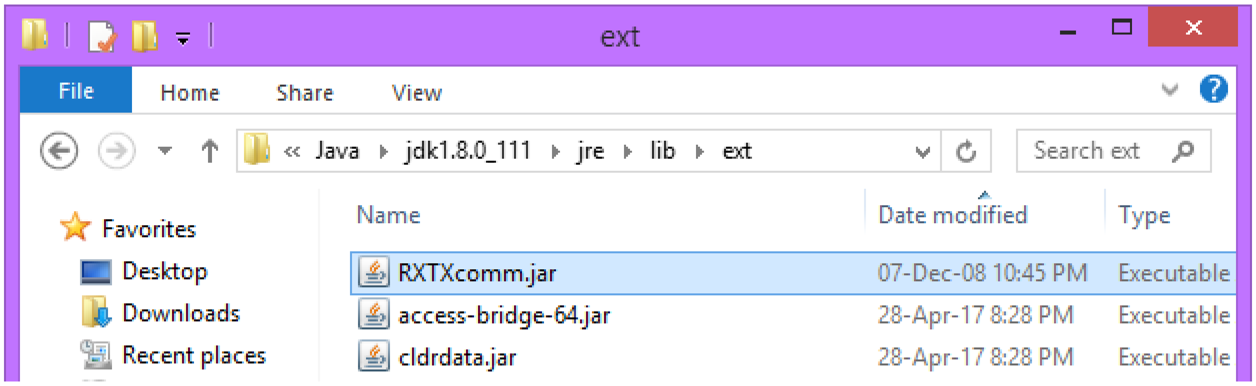
Name	Date modified	Type
RXTXcomm.jar	07-Dec-08 10:45 PM	Executable Jar File
dns_sd.jar	12-Aug-15 4:03 PM	Executable Jar File
access-bridge-64.jar	28-Apr-17 8:28 PM	Executable Jar File

5- Copy **rxtxSerial.dll** into >> **C:\Program Files\Java\<jdk-version>\jre\bin**



Name	Date modified	Type
rmiregistry.exe	28-Apr-17 8:28 PM	Application
rxtxSerial.dll	07-Dec-08 10:45 PM	Application exte
sawindbg.dll	28-Apr-17 8:28 PM	Application exte

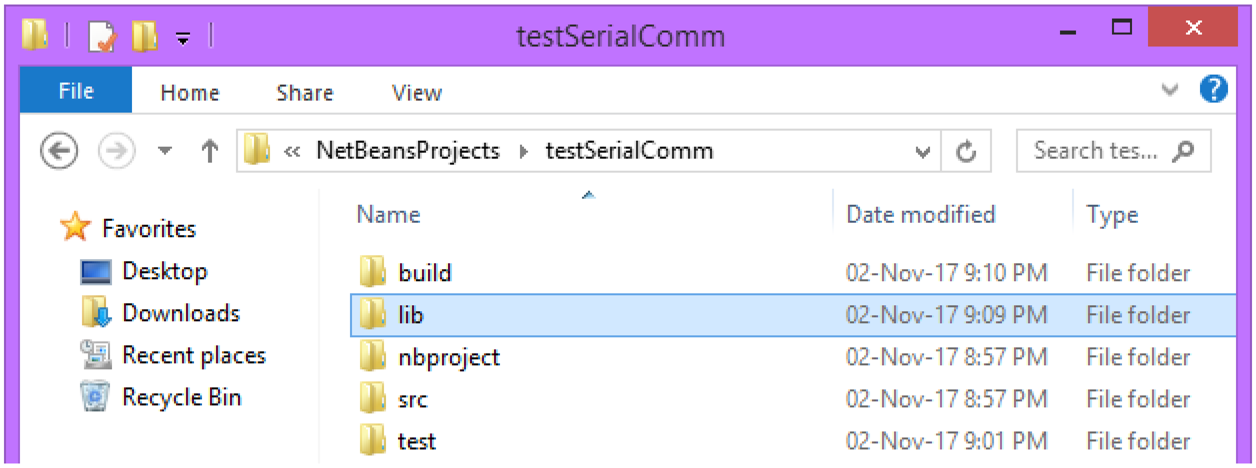
4- Copy **RXTXcomm.jar** into >> **C:\Program Files\Java\<jdk-version>\jre\lib\ext**



Name	Date modified	Type
RXTXcomm.jar	07-Dec-08 10:45 PM	Executable
access-bridge-64.jar	28-Apr-17 8:28 PM	Executable
cldrdata.jar	28-Apr-17 8:28 PM	Executable

7- Also copy **RXTXcomm.jar** into your project folder, inside a folder called lib

C:\Users\<PC-Name>\Documents\NetBeansProjects\<Project-Name>\lib

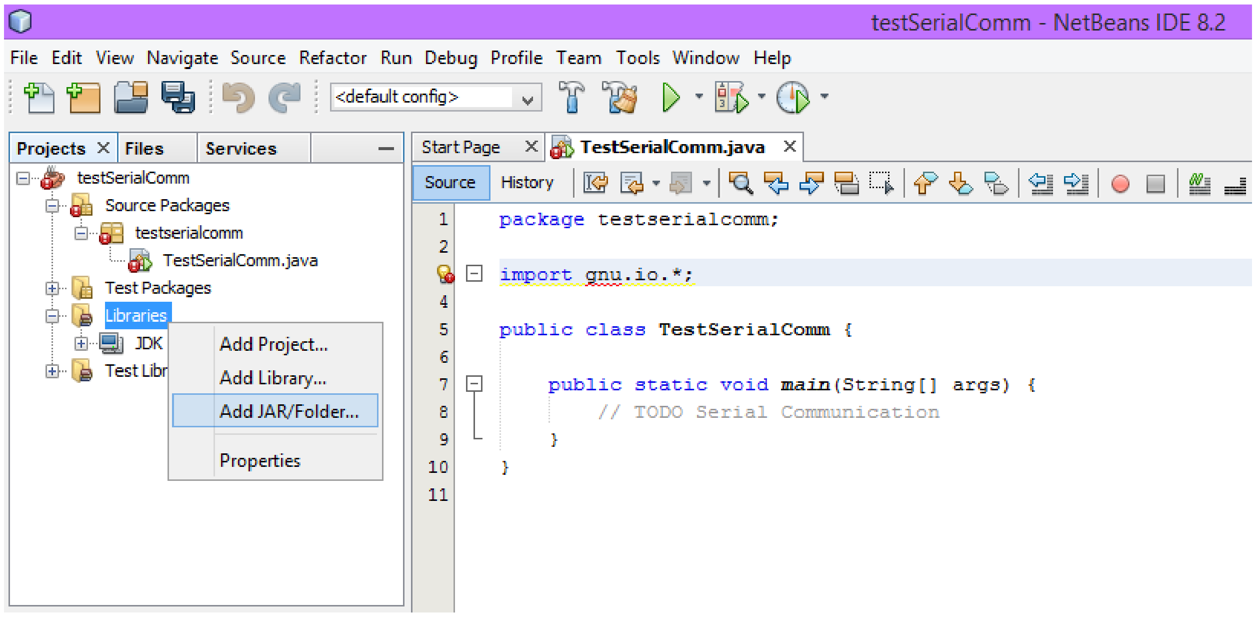


Name	Date modified	Type
build	02-Nov-17 9:10 PM	File folder
lib	02-Nov-17 9:09 PM	File folder
nbproject	02-Nov-17 8:57 PM	File folder
src	02-Nov-17 8:57 PM	File folder
test	02-Nov-17 9:01 PM	File folder

At every project, create the folder lib, and add **RXTXcomm.jar** in it.

04. How to add the library to your project

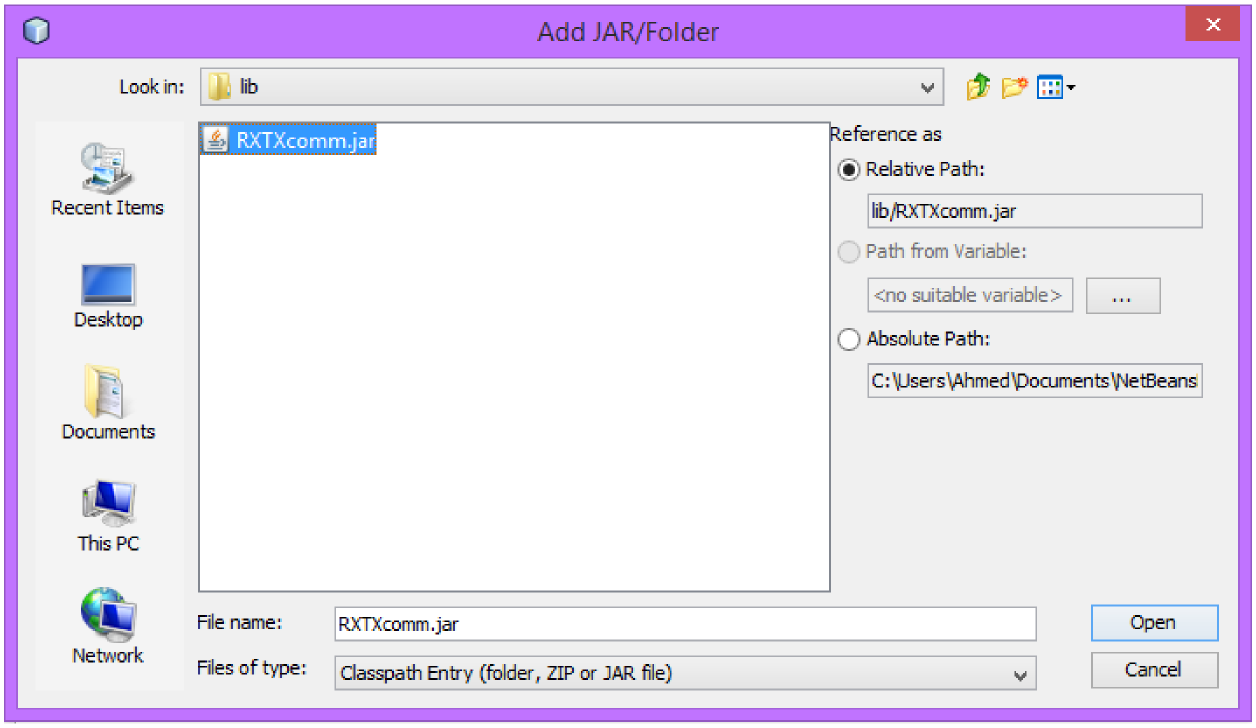
STEP-1



```java
package testserialcomm;

import gnu.io.*;

public class TestSerialComm {

    public static void main(String[] args) {
        // TODO Serial Communication
    }
}
```

STEP-2

Choose the file **RXTXcomm.jar** you've just added in the project folder under C:\Users\<PC-Name>\Documents\NetBeansProjects\<Project-Name>\lib



Look in: lib

RXTXcomm.jar

Reference as

(●) Relative Path: lib/RXTXcomm.jar

( ) Path from Variable: <no suitable variable> ...

( ) Absolute Path: C:\Users\Ahmed\Documents\NetBeans

File name: RXTXcomm.jar

Files of type: Classpath Entry (folder, ZIP or JAR file)

05. JavaFX, FXML and SceneBuilder

JavaFX is a bunch of packages which allows one to create rich internet and desktop applications. If you know Swing or AWT, then you know that they are used to create GUI applications. JavaFX also allows you to create GUI applications, but with less programming, and with more visual effects at your disposal.

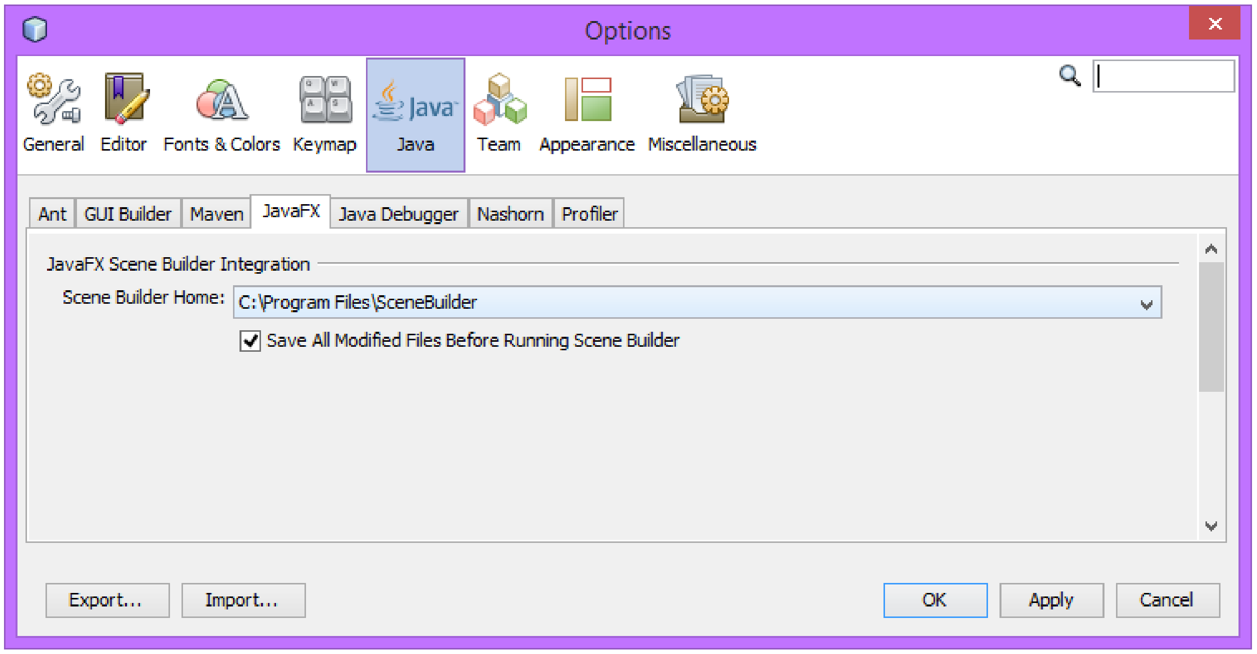
FXML is a file format which JavaFX uses to create the layout of screens, though you can even code your user interface directly. Although it's much easier to create FXML files using SceneBuilder.

SceneBuilder is an application where you can drag and drop JavaFX UI components, and then tell your JavaFX program to use the FXML file(s) to display the user interface.

Note that:

SceneBuilder is not part of the NetBeans IDE, it's a separate program so we'll have to download it and add it to the NetBeans to Open the FXML Documents.

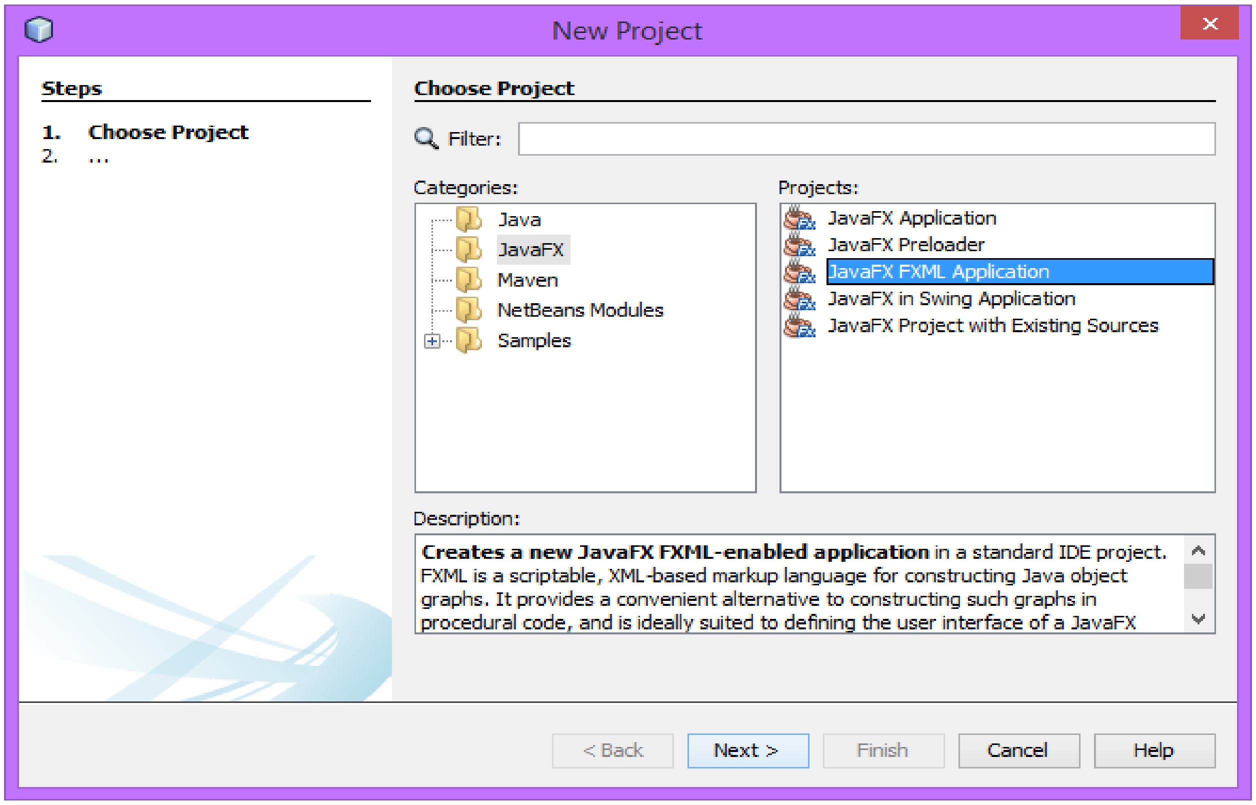
1. To download SceneBuilder Go to [http://gluonhq.com/products/scene-builder/]
2. Setup SceneBuilder on your machine
3. Add SceneBuilder to your NetBeans IDE
   from NetBeans Menu Bar [Tools -> Options]



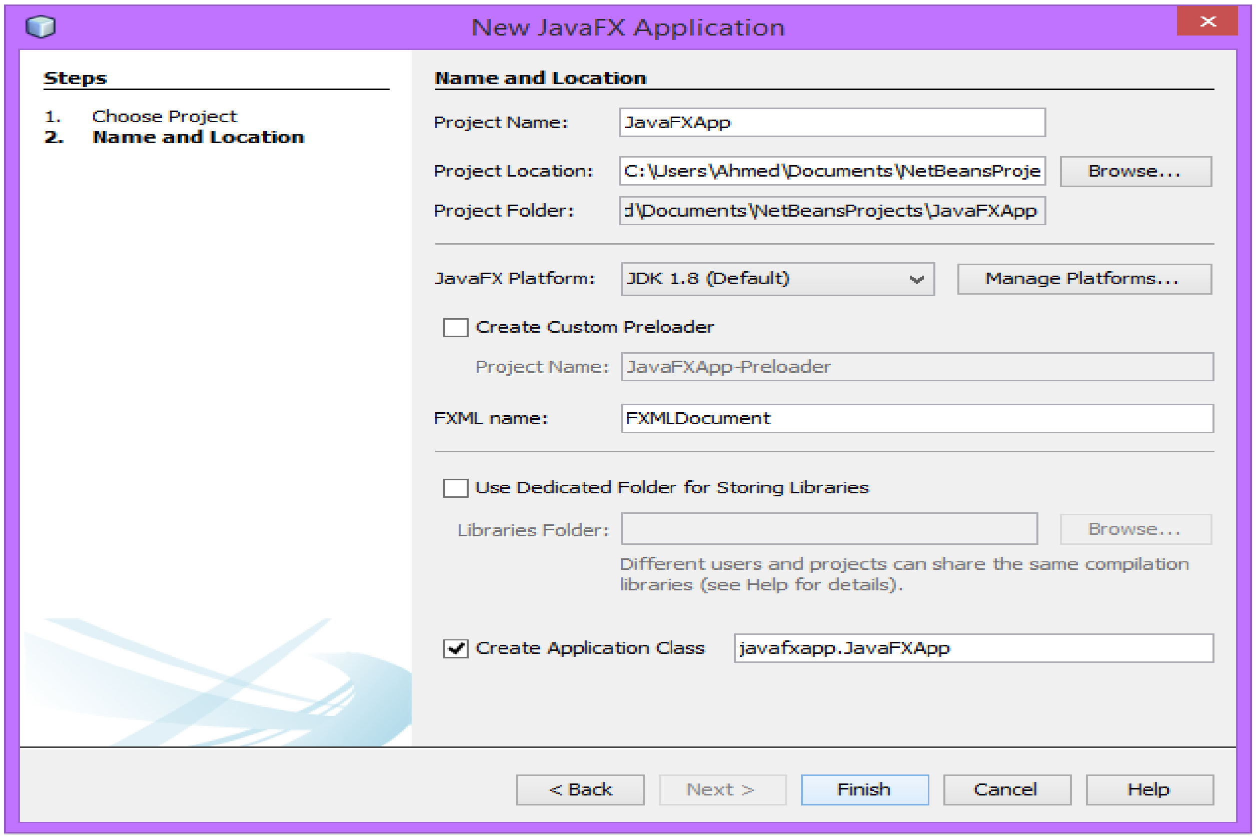
JavaFX Scene Builder Integration
Scene Builder Home: C:\Program Files\SceneBuilder
[✓] Save All Modified Files Before Running Scene Builder

on the drop down menu choose the location of Scene Builder Home, then click OK

06. Creating our first Java FXML Application



Creates a new JavaFX FXML-enabled application in a standard IDE project. FXML is a scriptable, XML-based markup language for constructing Java object graphs. It provides a convenient alternative to constructing such graphs in procedural code, and is ideally suited to defining the user interface of a JavaFX



Project Name: JavaFXApp

Project Location: C:\Users\Ahmed\Documents\NetBeansProje

Project Folder: d\Documents\NetBeansProjects\JavaFXApp

JavaFX Platform: JDK 1.8 (Default)

☐ Create Custom Preloader

Project Name: JavaFXApp-Preloader

FXML name: FXMLDocument

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

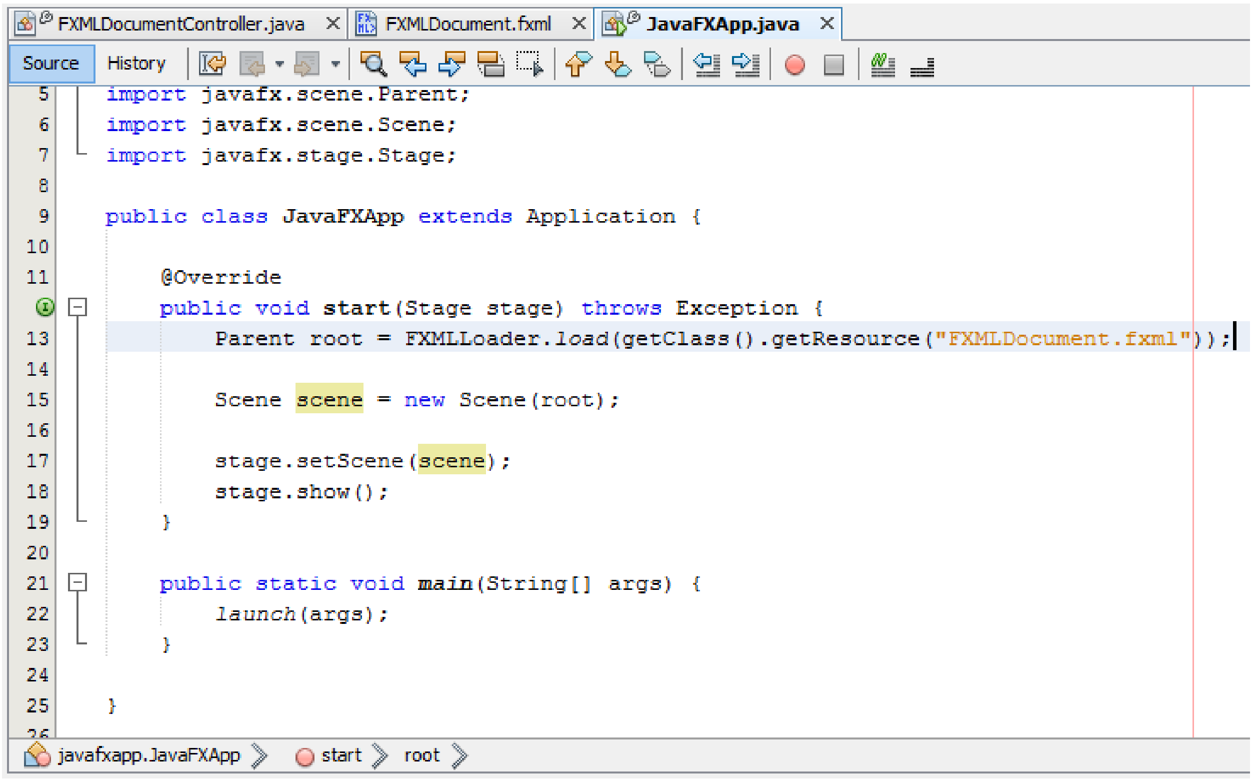
☑ Create Application Class javafxapp.JavaFXApp

After creating the project, you will end up with 3 files

• JavaFXApp.java	(the starting point of your application)
• FXMLDocument.fxml	(contains the FXML representation of the GUI)
• FXMLDocumentController.java	(for controlling the logic of the UI components)

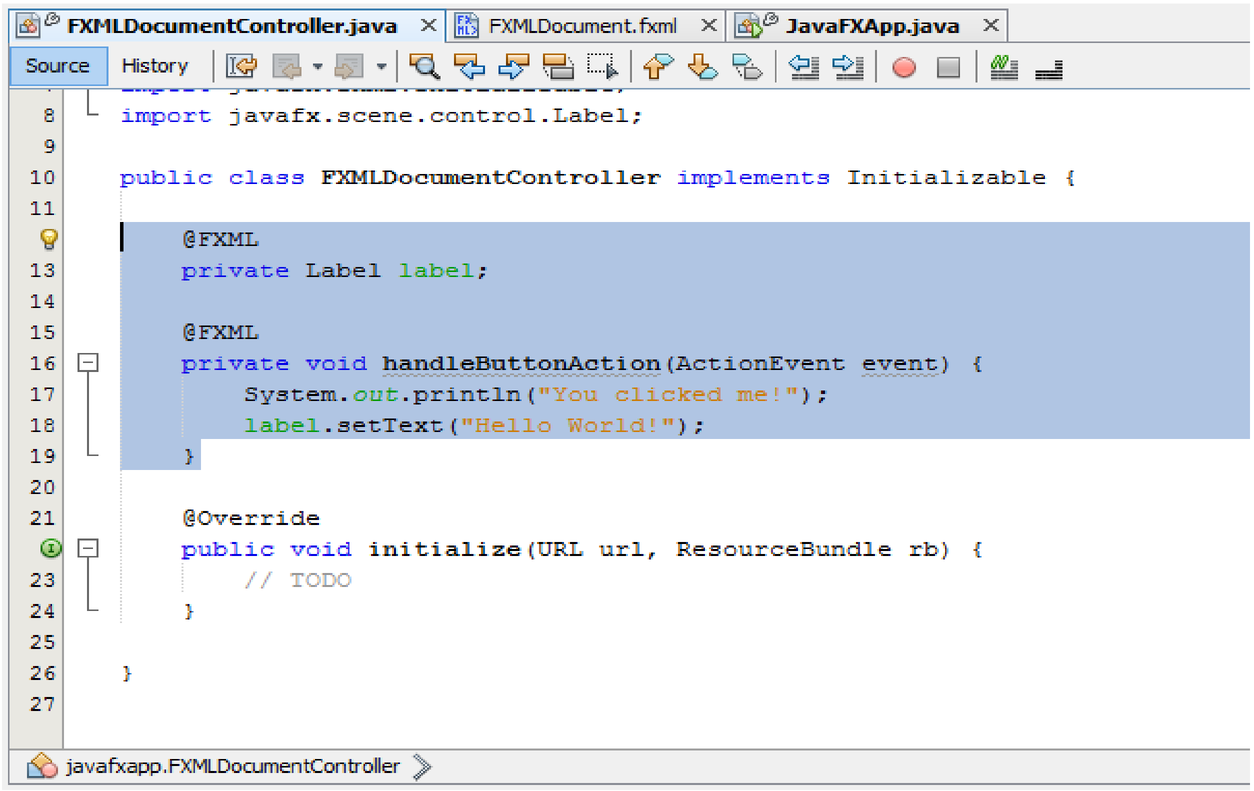
The project comes with a default template containing:

01. JavaFXApp.java File sets up the application starting from the fxml file.
we won't be modifying this file pretty much in our projects.



```java
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class JavaFXApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("FXMLDocument.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

02. FXMLDocumentController.java file contains the Logic of the UI.
we can remove the following code so we can put our own later.

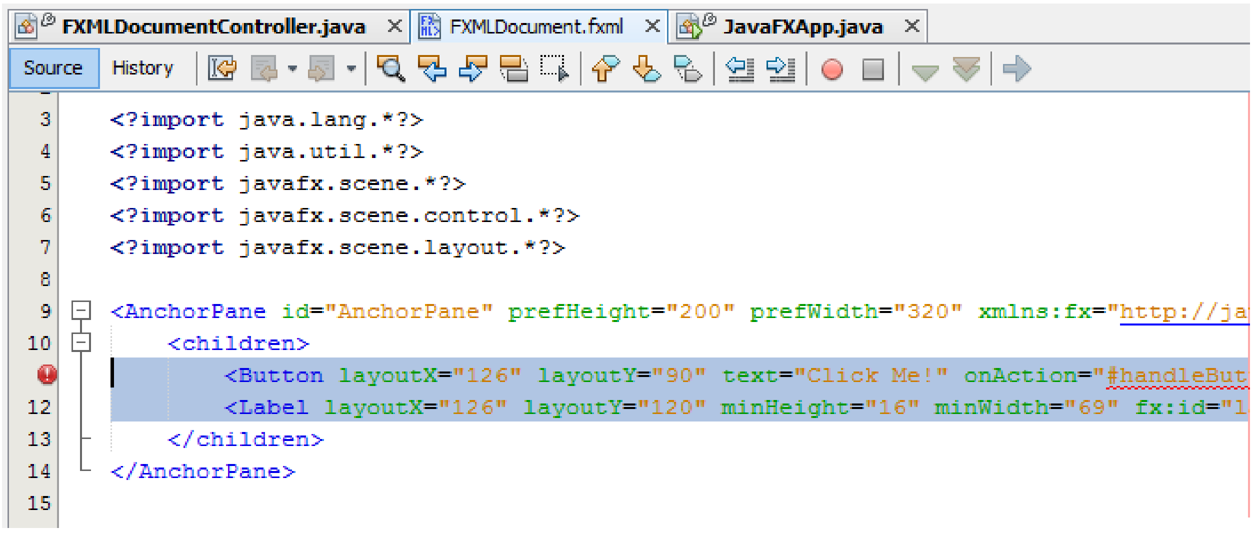

FXMLDocumentController.java FXMLDocument.fxml JavaFXApp.java
Source History

```java
import javafx.scene.control.Label;

public class FXMLDocumentController implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

javafxapp.FXMLDocumentController

The **initialize** method runs every time this FXML program File gets executed.
We will need it to setup and initialize things at the beginning of the application
and must be defined because FXMLDocumentController class is not abstract

03. FXMLDocument.fxml we will remove the fxml code that we just removed its logic and that's all we are going to do with this file.

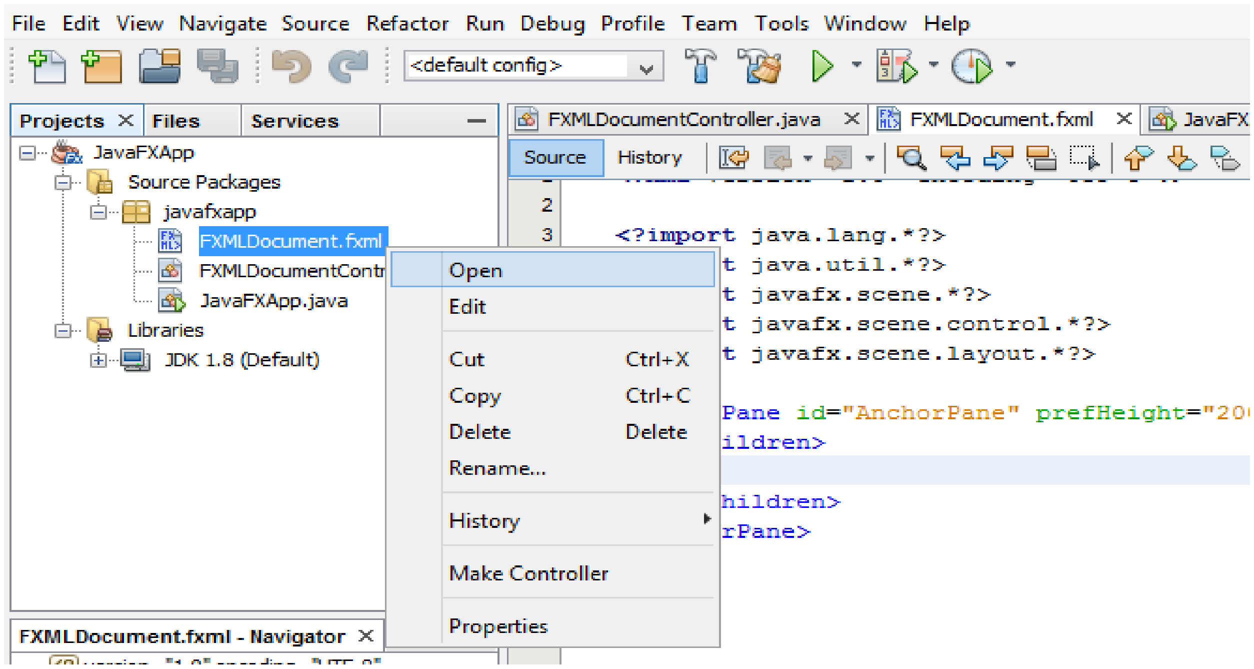

FXMLDocumentController.java FXMLDocument.fxml JavaFXApp.java
Source History

```xml
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://ja
    <children>
        <Button layoutX="126" layoutY="90" text="Click Me!" onAction="#handleBut
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="l
    </children>
</AnchorPane>
```

Now if we run the application it'll show an empty window, and that's fine for now.

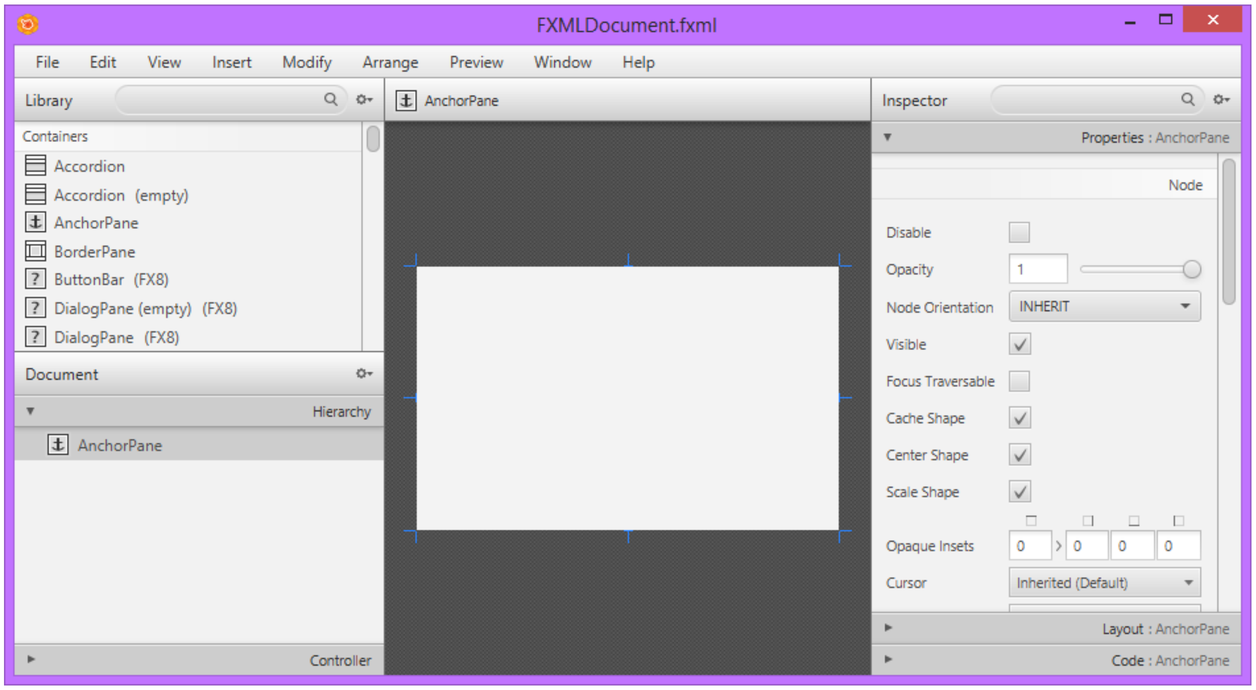
We won't be working with **FXMLDocument.fxml** file manually any more we will be using SceneBuilder a FXML document designer with a drag-and-drop User Interface.

If your setup of SceneBuilder in your NetBeans IDE as in Step-5
When you open the fxml document like the following



```
<?import java.lang.*?>
t java.util.*?>
t javafx.scene.*?>
t javafx.scene.control.*?>
t javafx.scene.layout.*?>

Pane id="AnchorPane" prefHeight="20
ildren>

hildren>
rPane>
```

This will open the **FXMLDocument.fxml** file **in the SceneBuilder,** then drag-and-drop elements



Disable	☐
Opacity	1
Node Orientation	INHERIT
Visible	✓
Focus Traversable	☐
Cache Shape	✓
Center Shape	✓
Scale Shape	✓
Opaque Insets	0 0 0 0
Cursor	Inherited (Default)

Using RXTX SerialPort Class

```
// step: 1- Creating a port
CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier("COM1");
CommPort commPort = portIdentifier.open("AppName",2000);

// step: 2- Making it a Serial Port, and setting its parameters
SerialPort serialPort = (SerialPort) commPort;
serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);

// step: 3- Gaining access to the i/o streams
InputStream in = serialPort.getInputStream();
OutputStream out = serialPort.getOutputStream();

// step: 4- Sending data through the stream
out.write('r');

// step: 5- Closing the Streams
in.close();
out.close();

// step: 6- Closing the Serial Port
serialPort.close();
```

SerialPort Class is an abstract class and cannot be instantiated on its own

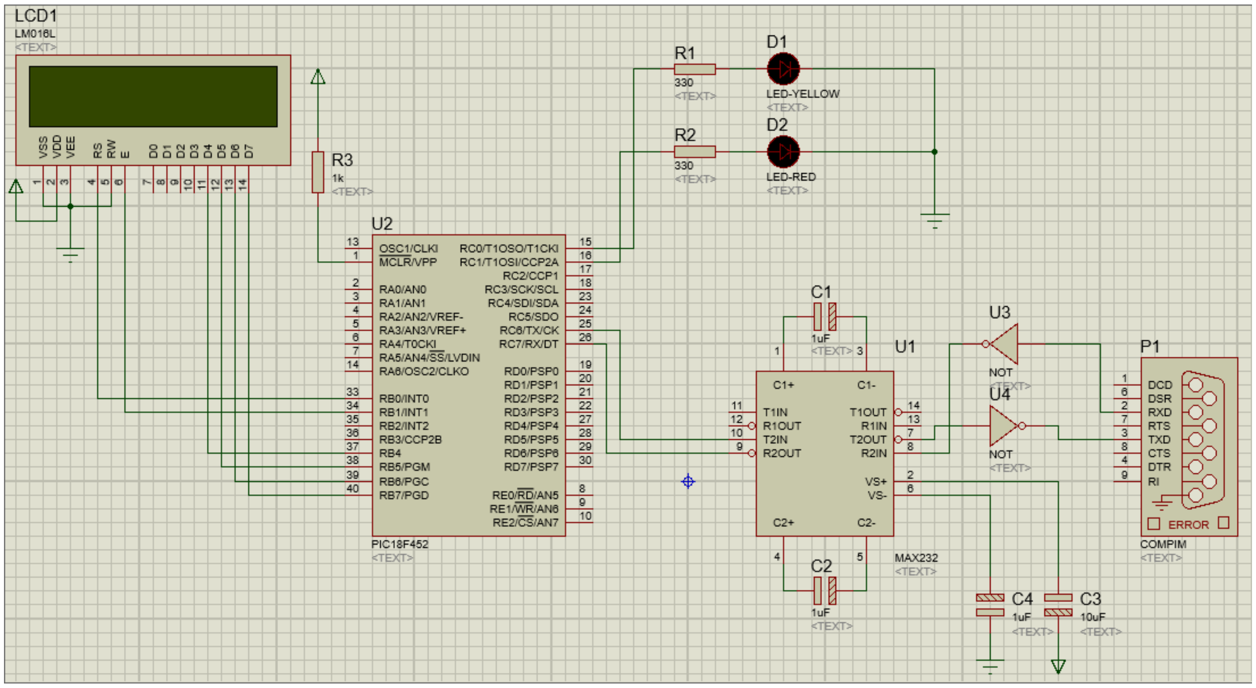
So, we created a communication port then Casted it to a serial port

The serial port class works only through streams and doesn’t contain any other premade methods to handle sending and receiving data.

Unlike the Exceptions in C#, in Java we have compile-time exceptions that you must catch or your program won’t compile.

Now that we learned the small building blocks, let's use them to create our Serial Communication Examples.

Example 1: interfacing PC with PIC microcontroller using serial communication



MicroC PIC Code

```
// LCD Pins-Setup Code Goes Here

unsigned char Recv;

void main() {
   TRISA = TRISB = TRISD = TRISC = 0;
   TRISC.F7 = 1;

   ADCON1 = 0X07;

   Lcd_Init();                          // Initialize LCD
   Lcd_Cmd(_LCD_CLEAR);                 // Clear display
   Lcd_Cmd(_LCD_CURSOR_OFF);            // Cursor off
   Lcd_Out(1,2,"Serial Comm");          // Write text in first row

   Uart1_Init(9600);
   Delay_ms(100);

   for(;;) {
      if(UART1_Data_Ready()) {
         Recv = UART1_Read();
      }

      if(Recv == 'Y' || Recv == 'y') {
         PORTC.F0 = 1;
         PORTC.F1 = 0;
         Lcd_Cmd(_LCD_CLEAR);
      }
      else if(Recv == 'R' || Recv == 'r') {
         PORTC.F0 = 0;
         PORTC.F1 = 1;
         Lcd_Cmd(_LCD_CLEAR);
      }
      else if(Recv == 'L' || Recv == 'l') {
         PORTC.F0 = 0;
         PORTC.F1 = 0;
         Lcd_Out(2, 1, "Test LCD Output");
      }

      Delay_ms(20);
   }
}
```

Writing the Java Serial Communication Program

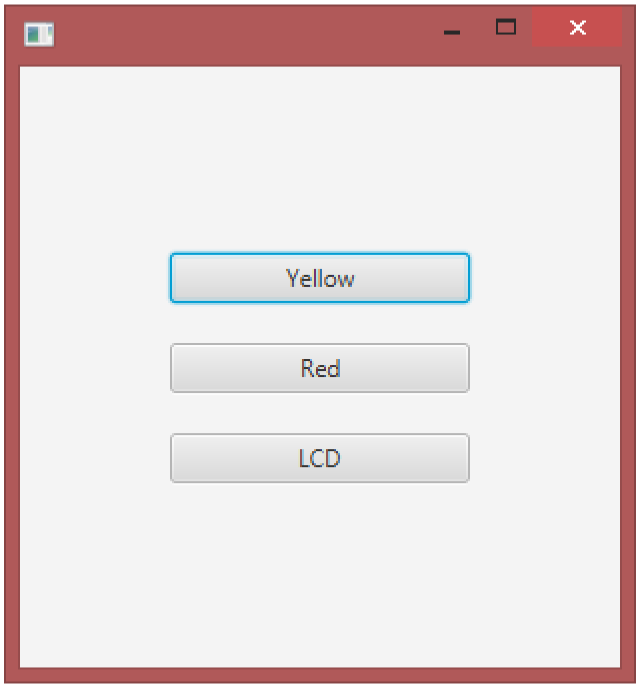
Step-1 Create a Java FXML Application <like we did in Section-06>

Step-2 Add the RXTXcomm.jar to your Project <like we did in Section-04>

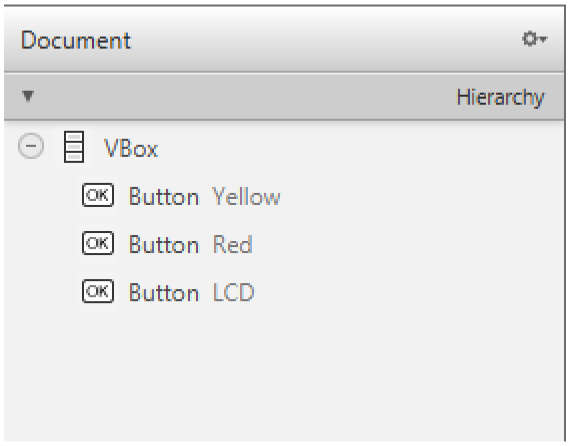
Step-3 Designing the Java FXML User Interface (using drag-and-drop)

Step-4 Writing the Java FXML Controller Code

GUI <using Scene Builder Drag-and-Drop Elements>



Components <VBox and 3 Buttons>



Java Controller Code

```
public void YellowButtonClicked() {

    try {

            String portName = "COM1";
            CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(portName);
            CommPort commPort = portIdentifier.open(this.getClass().getName(),2000);

            SerialPort serialPort = (SerialPort) commPort;
            serialPort.setSerialPortParams( 9600, SerialPort.DATABITS_8, SerialPort.STOPBITS_1,
                                                    SerialPort.PARITY_NONE );

            InputStream in = serialPort.getInputStream();
            OutputStream out = serialPort.getOutputStream();

            out.write('y');

            in.close();
            out.close();

            serialPort.close();

        } catch (PortInUseException e) {
            e.printStackTrace();
        } catch (UnsupportedCommOperationException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (NoSuchPortException e) {
            e.printStackTrace();
        }

}
```

The controller methods YellowButtonClicked, RedButtonClicked, LcdButtonClicked are all the same code except **out.write('y');** which contains the letter to be sent.

Learning Materials

For interested readers

In this appendix we gave you an up and running introduction on how to get started with Serial Communication in java but there is more to it than this, so here we provide you with a list of resources and materials for extensive learning on the subject.

Learning the Java Programming Language, we recommend
Oracle Tutorials [https://docs.oracle.com/javase/tutorial/]

Learning the JavaFX and Java FXML and Using SceneBuilder, we recommend
Oracle Tutorials [https://docs.oracle.com/javase/8/javase-clienttechnologies.htm]

Learning and using the RXTX library, we recommend
RXTX Wiki [http://rxtx.qbang.org/wiki/index.php/Main_Page]
Oracle Docs [
https://docs.oracle.com/cd/E17802_01/products/products/javacomm/reference/api/index.html]