



*Back in 2013, our development team created a dashboard for Android which could display electricity network data in a workshop. Their main problem was that the device that collected the data stood alone and had to be connected to an Android device via a serial port. One of the developers published a blog post which explored the possibility of using a COM port in applications for an Android OS. It has soon turned out that many people around the globe have come across similar issues when trying to connect the devices and, as a result, the post has gained much popularity since the date of its publication. As this post is quite popular on our blog, we think that it is high time we updated the post to make it easily available for anyone who has recently had any difficulties with COM ports for an Android device.*

A few months ago I was asked to [develop a dashboard for Android](#) which could display electricity network data in a workshop. The main problem was that the device that collects the data stands alone and must be connected to an Android device via a serial port. In this article, I will explore the possibility using a COM port in [applications for an Android OS](#). It is needed when your application must receive/send data from the other device that only has RS 232 serial port as a means of communication.

## What do we need?

- First, you need to download Android NDK to work with native code in Java: <http://developer.android.com/tools/sdk/ndk/index.html> It will be necessary to call code on C to work with the serial port.
- Second, it is necessary to check out the source code of the project from [svn](#) to work with Serial port RS 232: android-serialport-api

## Organization of the project

It is necessary for us to create a **/jni folder** and to copy the contents of the project /jni folder, that was downloaded from svn (or simply to copy complete folder /jni), in the Android project. After that we need to add the following files from the downloaded project:

- SerialPort.java
- Application.java
- SerialPortActivity.java
- SerialPortFinder.java

The given files allow us to work with the COM port but you will need to edit the files to serve the needs of the project. It is known all devices in the systems are similar to a \*nix one are on a **path /dev**. To establish a path to the device (COM port) and baud rate it is necessary to set the appropriate values of the fields in the Application.java file:

```
String          path          =          -path          to          device-;
int <g class="gr_ gr_98 gr-alert gr_spell gr_run_anim ContextualSpelling
ins-del multiReplace" id="98" data-gr-id="98">baudrate</g> = -<g class="gr_
```

Class **SerialPortActivity.java** is an extension of Activity class and contains an abstract method protected abstract void **onDataReceived** (final byte [] buffer, final int size). You can inherit the Activity from this class where they will work with the serial port, and to process data acquisition from the port having redefined the method onDataReceived.

```
@Override
protected void onDataReceived(final byte[] buffer, final int size) {
runOnUiThread(new Runnable()
```

It is possible to extract logic of the information system port in your own class/classes and not to use the inheritance from SerialPortActivity.java. You can receive the lists of all the devices and their paths by means of class **SerialPortFinder.java** with its methods **getAllDevices ()** and **getAllDevicesPath ()** accordingly. The write-In port is carried out by using a simple record in OutputStream created by means of class SerialPort.java, COM port.

```
<g class="gr_ gr_93 gr-alert gr_spell gr_inline_cards
gr_disable_anim_appear ContextualSpelling ins-del multiReplace" id="93"
data-gr-id="93">mOutputStre</g>am.write(new
```

$$\begin{aligned} & \tilde{A}\tilde{c}\hat{a}, -\tilde{A}; \tilde{A}, \hat{A}-\tilde{A}f\hat{a}\in|\tilde{A}, \hat{A}; \tilde{A}f\tilde{A}'\tilde{A}+\hat{a}\in^{\text{TM}}\tilde{A}f\hat{A}\tilde{c}\hat{A}\in\tilde{S}\hat{A}-\tilde{A}...\hat{A}; \tilde{A}f\tilde{A}'\tilde{A}\hat{c}\hat{a}, -\tilde{A}; \tilde{A}f\hat{a}\in\tilde{S}\hat{A}, \hat{A}\tilde{c}\tilde{A}f\tilde{A}'\tilde{A}+\hat{a}\in^{\text{TM}}\tilde{A}f \\ & \hat{a}\in\tilde{A}\tilde{c}\hat{a}, -\hat{a}, \tilde{c}\tilde{A}f\tilde{A}'\tilde{A}\hat{c}\hat{a}, -\tilde{A}f\hat{A}\tilde{c}\hat{A}\tilde{c}\hat{A}\in\tilde{S}\hat{A}-\tilde{A}\tilde{c}\hat{a}\in\tilde{Z}\hat{A}\tilde{c}\tilde{A}f\tilde{A}'\tilde{A}+\hat{a}\in^{\text{TM}}\tilde{A}f\hat{A}\tilde{c}\hat{A}\tilde{c}\hat{A}\in\tilde{S}\hat{A}-\tilde{A}...\hat{A}; \tilde{A}f\tilde{A}'\tilde{A}\hat{c}\hat{a}, -\tilde{A}; \tilde{A} \\ & f\hat{a}\in\tilde{S}\hat{A}, \hat{A}\tilde{c}\tilde{A}f\tilde{A}'\tilde{A}+\hat{a}\in^{\text{TM}}\tilde{A}f\hat{a}\in\tilde{A}\tilde{c}\hat{a}, -\hat{a}, \tilde{c}\tilde{A}f\tilde{A}'\tilde{A}\hat{c}\hat{a}, -\tilde{A}; \tilde{A}f\hat{a}\in\tilde{S}\hat{A}, \hat{A}\tilde{c}\tilde{A}f\tilde{A}'\tilde{A}+\hat{a}\in^{\text{TM}}\tilde{A}f\hat{a}\in\tilde{S}\hat{A}, \hat{A}\tilde{c}\tilde{A}f\tilde{A}'\tilde{A}, \hat{A} \end{aligned}$$

```
( "text" ).getBytes( ) ;
```

## Accessing a serial port

Native code loading in the Android application may be found in class `SerialPort.java` by calling **System.loadLibrary** ( ' serial\_port ' ):

```
private native static FileDescriptor <g class="gr_ gr_101 gr-alert gr_gramm
gr_run_anim Grammar multiReplace" id="101" data-gr-id="101">open</g>(String
path, int <g class="gr_ gr_115 gr-alert gr_spell gr_run_anim
```

The parameter **serial\_port** is a module that has occurred as a result of code C compilation through/via NDK. It is specified in file make /jni/Android.mk. File SerialPort.c in /jni folder and contains native functions of the system calls to work with Serial port. Such parameters for the COM port as Data bits, Parity, Stop bits and the others may be changed in this file by means of structure termios, for example:

|             |           |          |
|-------------|-----------|----------|
| cfg.c_cflag | =         | ~PARENB; |
| cfg.c_cflag | &~CSTOPB; |          |
| cfg.c_cflag | &~CSIZE;  |          |

This information about **termios.h** may be found under the following links:  
<http://ulisse.elettra.trieste.it/services/doc/serial/config.html>  
<http://pubs.opengroup.org/onlinepubs/007908775/xsh/termios.h.html>

After changing the file SerialPort.c it is necessary to compile the libraries as follows:

- Open command line
- Go to the NDK folder
- Set path to the Android project – set `NDK_PROJECT_PATH=` -path to your android project-
- Run – `ndk-build`

The libraries will be compiled and added in a /libs folder of your project. After that it is possible to perform the following command in order to install an Android application in a device:

```
<g class="gr_ gr_120 gr-alert gr_spell gr_run_anim ContextualSpelling ins-del multiReplace" id="120" data-gr-id="120">adb</g> install &lt;path to  
your .apk file&gt;
```

## Result

As a result we have all the instruments that can help us to connect a third-party device via a serial port. Sending and receiving data using these instruments also allow us to manage devices which give us great opportunities for new business ideas.