Recibido: 15 de setiembre del 2016
Aprobado: 20 de octubre del 2016

Aplicativo Android *datalogger* serial-USB para sistemas embebidos

Nícolas Francisco Figueroa Mosquera
nicolas.figueroa@cip.org.pe
NFM Robotics E.I.R.L. Lima, Perú

Resumen

El presente artículo muestra el aplicativo realizado en Android que permite registrar los datos adquiridos por comunicación serial de cualquier sistema embebido; así como probar las entradas y salidas en un equipo portátil, como lo es un dispositivo móvil con sistema operativo Android. Para su realización se utilizaron como referencia las librerías de Ohwada, las cuales se modificaron para tener la opción de registrador en la memoria SD del dispositivo móvil desde el momento que uno presiona el botón iniciar hasta que uno presiona el botón parar. Además, la forma de mostrar los valores adquiridos por serial en una sola consola fue modificada, junto con algunos parámetros de configuración en la transmisión de datos.

Palabras clave: sistema embebido / comunicación serial / registrador / monitor / verificador de datos

Abstract

This article shows the application made in Android that enables the registration of the data obtained by serial communication from any embedded system, as well as the testing of the inputs and outputs in a portable device equipped with the Android operating system. To achieve this, the Ohwada libraries were used as reference, modified so that the register option in the SD memory of the mobile device was available from the moment the start button is pressed until the moment the stop button is pressed. The way of showing the acquired values by serial in a single console was also modified, along with some configurations parameters in the transmission of data.

Keywords: embedded system / serial communication / data logger / monitor / data verifier

1. Introducción

1.1. Motivación

Los estudiantes de las carreras de Ingeniería suelen cursar asignaturas que involucran el aprendizaje de sistemas embebidos, tales como: programación para ingenieros, arquitectura de computadoras, microcontroladores y robótica, innovación tecnológica, entre otras; por lo cual les sería muy útil poseer una herramienta implementada en su dispositivo móvil que haga las veces de *datalogger*.

1.2. Planteamiento del problema

En la mayoría de estas asignaturas, cuando se aborda el tema de sensores analógicos, proceden a verificar los datos adquiridos mediante una conexión con una computadora de escritorio o con una *laptop*, pero existen situaciones donde el sistema embebido se encuentra integrado a un robot móvil y resulta poco práctica la conexión cableada a la computadora para la toma de datos.

1.3. Aportes

- Esta aplicación sería aplicable a cualquier sistema embebido que maneje una comunicación serial-USB como Arduino y Raspberry.
- La ventaja de usar una comunicación serial-USB es la no dependencia de una conexión wifi o de otra conexión inalámbrica que implique una fuente de alimentación externa para ambos dispositivos. Es por ello que se usaría la misma conexión serial-USB para que el dispositivo móvil brinde la fuente de alimentación al sistema embebido.

2. Objetivos de estudio

2.1. Objetivo general

Diseñar y elaborar una aplicación intuitiva para dispositivos móviles, que permita la comunicación serial-USB y el almacenamiento de dichos datos en la memoria del dispositivo móvil para un posterior análisis.

2.2. Objetivos específicos

i. Programar en java bajo el entorno de Eclipse una aplicación en Android.

ii. Establecer la comunicación unidireccional y bidireccional entre la aplicación y un sistema embebido como Arduino.

3. Fases de desarrollo

Las fases que involucran el desarrollo del proyecto van desde la concepción hasta la realización de la misma, las cuales son las siguientes:

i. *Concepción de funcionalidades e interfaz*

En esta fase se establece las funcionalidades que tendrá la aplicación, además de realizar la distribución de los botones, indicadores y ventanas de tal forma que le sea útil y práctico para quien lo maneje.

ii. *Referencia de aplicaciones anteriores y disponibilidad de código*

Esta segunda fase involucra la búsqueda de referencias anteriores, como programas similares que sean de código abierto y puedan ser tomarlos de referencia para el desarrollo de la aplicación.

iii. *Pruebas de comunicación*

Se realizan pruebas de comunicación entre la aplicación ya descargada en el dispositivo móvil y un sistema embebido, verificando si se puede enviar y recibir datos sin ningún problema además de almacenarlos en la memoria.

iv. *Desarrollo de la interfaz gráfica*

Una vez lograda una comunicación bidireccional entre el sistema embebido y la aplicación, se procede a mejorar la interfaz gráfica aplicando el resultado de la fase 1.

v. *Implementación y pruebas finales*

En esta fase se hacen las pruebas finales con la interfaz gráfica ya terminada, verificando el correcto funcionamiento de la aplicación.

4. Resultados

A continuación, se muestra los resultados obtenidos en la realización del proyecto.

4.1. Interfaz gráfica

En la interfaz principal, la cual se puede observar en la figura 1, una vez conectado el cable de comunicación el mensaje “no conectado” cambiará por la dirección reconocida del terminal micro USB. Se pueden ingresar la cantidad de baudios, el cual por defecto se encuentra en 9 600. Este valor se cambia una vez ingresado la cantidad deseada luego de pulsar el botón *baudios*.

Se posee la opción de poder enviar datos, escribiendo los caracteres deseados y luego pulsando el botón *enviar*.

Figura 1. Layout principal

Elaboración propia

Si se está transmitiendo datos desde un inicio, estos se verán reflejados en el *TextView* a manera de consola, mostrando siempre los últimos datos ingresados con *autoscroll*.

En el momento que se desee registrar los datos se pulsa el botón *Iniciar*, con ello el color de letra cambiará a color verde indicando que se está realizando el almacenamiento de datos a una variable tipo *string*. Cuando se desee parar, se pulsa el mismo botón. Luego solo se tiene que escribir el nombre del archivo el cual puede ser alfanumérico y se presiona el botón "Grabar". Automáticamente será grabado en la memoria SD del celular.

El segundo *layout* es *acerca*, figura 02, el cual solo representa una pequeña descripción de la funcionalidad de la aplicación y su creación.

Figura 2. Layout acerca

Elaboración propia

4.2. Elección de las librerías

Las librerías que se usaron para la comunicación serial-USB son de Ohwada (2014), el cual ofrece las siguientes clases java:

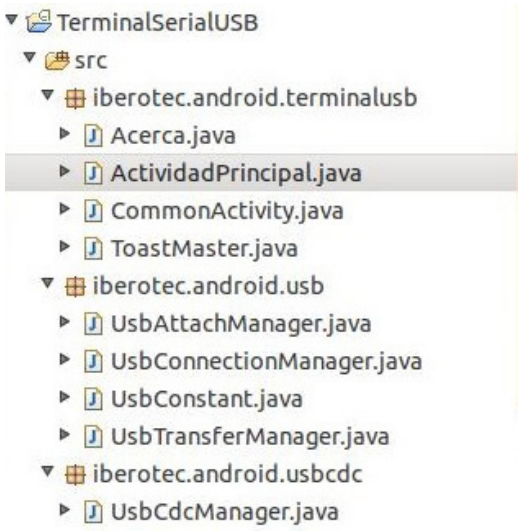
- Common Activity
- Toast Master
- Usb Attach Manager
- Usb Connection Manager
- Usb Constant
- Usb TransferManager
- Usb Cdc Manager

Estas clases dieron buenos resultados al ejecutar un ejemplo predeterminado, demostrando su funcionalidad en la comunicación serial-USB.

4.3. Lógica de ejecución

Las clases en Java que se utilizaron se pueden apreciar en la figura 3:

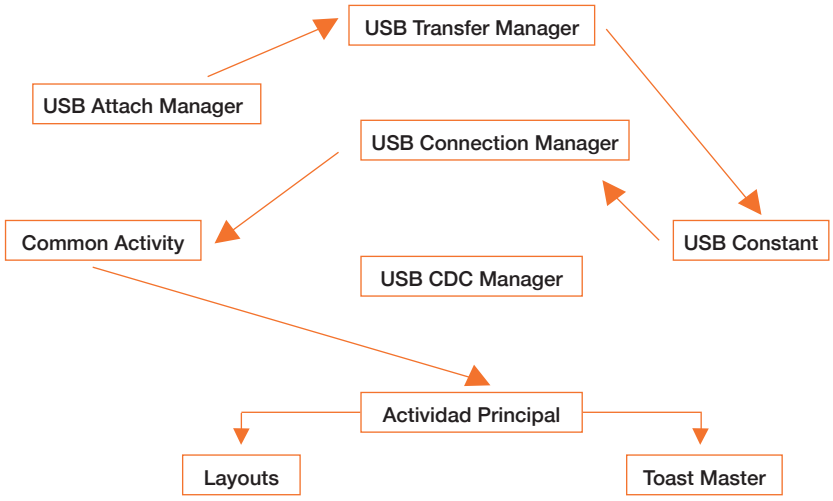
Figura 3. Clases en Java utilizadas



Elaboración propia

Las interacciones entre las clases utilizadas se pueden representar de la siguiente forma:

Figura 4. Interacción de clases serial-USB



Elaboración propia

Una vez establecida la conexión, el Usb Attach Manager establece el permiso de registro y establece la conexión, además de notificar algún error en la conexión. Luego

el USB Connection Manager es donde se obtiene el ID del producto y del vendedor; como se sabe, todo dispositivo móvil con conexión USB posee un identificador del tipo 16c0:05df, donde los primeros cuatro valores son el vendedor y los subsiguientes después de los dos puntos representan el producto. Estos datos son muy importantes para establecer una comunicación USB.

El USB CdC Manager, USB Transfer Manager y USB Constant se encargan de establecer los parámetros de configuración, tamaño del buffer, tiempo de recepción, tiempo muerto de datos, obtención de datos en un arreglo de *bytes* y convertirlos a *string*, para luego ser invocados en la actividad principal.

El Common Activity, establece las configuraciones modificables por el usuario de acuerdo al tipo de conexión; son los datos por defecto que carga el *layout*. Estos datos son cargados en la actividad principal, que utiliza la recepción y el envío de datos y la muestra en pantalla mediante el uso del *layout*. Para el envío de notificaciones como de conexión y desconexión, entre otros, se hace uso del Toast Master.

4.4. Modificación de parámetros de transmisión USB

Al momento de programar en Java, uno de los factores más importantes al usar librerías son los permisos que permiten ejecutar algunas funciones del dispositivo móvil. En este caso se debe brindar el permiso de habilitar la función de escritura en la memoria SD; para lograr ello, de acuerdo al tutorial "13-Almacenamiento de datos en un archivo de texto localizado en una tarjeta SD" (s.f.), se escribe el permiso Write_External_Storage.

Figura 5. Permiso de escritura en la memoria SD del dispositivo móvil

```
<uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="22" />

<uses-feature android:name="android.hardware.usb.host" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<application
```

Elaboración propia

En la actividad principal se decidió utilizar un *TextView* en el *layout* a modo de consola para la visualización de caracteres; dentro de la actividad principal se utilizó una variable llamada *Log* la cual almacena los datos a mostrar. Una de las características de la consola es el de mostrar los últimos datos en pantalla, para ello se creó la función *public static void Log*. Habilitando la opción de *scrolling* mediante la siguiente línea de comando: *log.setMovementMethod* (*new Scrolling Movement Method*()).

Figura 6. Programa para la creación de la consola de datos recibidos

```
@Override
protected void onCreate( Bundle savedInstanceState ) {
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_main );
    initManager();
    initView();

    mEditTextBaudrate = (EditText) findViewById( R.id.EditText_baudrate);
    mEditTextSend = (EditText) findViewById( R.id.EditText_send );

    log = (TextView)findViewById(R.id.log);
    log.setMovementMethod(new ScrollingMovementMethod());
    btiniciar= (Button) findViewById(R.id.iniciar);
    et1 = (EditText) findViewById(R.id.EditText_nombrearchivo);
    indicador = (ImageView) findViewById(R.id.indicadorled);

    public static void Log (String string) {
        if (log !=null) {
            log.append(string);
            final Layout lay = log.getLayout();
            if (lay != null) {
                int diferencia = lay.getLineBottom(log.getLineCount() - 1)
                        - log.getScrollY() - log.getHeight();
                if (diferencia > 0) {log.scrollBy(0,  diferencia);}
            }
        }
    }
```

Elaboración propia

Dentro de la clase Usb Transfer Manager, se modificó los parámetros de ampliar el tamaño del *buffer* al recibir el *byte,* pero manteniendo el tiempo muerto y el tiempo de espera del dato, pues ello no afectaba en la consola Log creada. Inicialmente existían problemas al considerar un arreglo para mostrar los datos en pantalla, que es la forma por defecto que las librerías de Ohwada (2014) ofrecían, donde los datos se mostraban cortados por más que se modificaran los parámetros de tiempo muerto y tiempo de espera. Es por ello que se cambió por una consola que muestre todos los datos a partir de una variable.

Figura 7. Código de configuración de transferencia USB

```
public class UsbTransferManager extends UsbConnectionManager {

    // send
    private final static int SEND_BULK_TIMEOUT = 10;

    // recv
    private static final int RECV_BYTE_BUFFER_SIZE  = 1024;//256
    private static final int RECV_BLUK_TIMEOUT = 50;//50
    private static final long RECV_BLUK_WAIT = 10;//10

    // recv
    private byte[] mRecvByteBuffer = new byte[ RECV_BYTE_BUFFER_SIZE ];
    private String mRecvStringBuff = "";
    private boolean isRecvRunning = false;

    // callback
    private OnRecieveListener mRecieveListener = null;
```

Elaboración propia

Dentro de la clase Common Activity se consideró 9600 baudios, con un largo de caracteres de 1024, y una espera máxima de 100 ms.

Figura 8. Código de configuración de caracteres recibidos

```
public class CommonActivity extends Activity {

    // debug
    protected final static boolean D = true;
    private final static String TAG = "UsbSerial";

    // serial
    private final static int BAUDRATE = 9600;

    // recieve
    private final static int MAX_STRING_LENGTH = 1024;
    private final static int MAX_STRING_WAIT = 100;

    // char
    protected final static String SPACE = " ";
    private final static String LF = "\n";

    // object
    private UsbCdcManager mUsbCdcManager;

    // UI
    private TextView mTextViewConnect;
```

Elaboración propia

4.5. Registro de datos

Para el registro de datos se creó una variable llamada *grabando* la cual se usa de referencia. Para el caso de no registrar datos se cambia el texto en pantalla por *iniciar* mediante un *set Text* y se cambia la imagen por *off* mediante un *set Image Resource*. En caso contrario se cambia por *parar* y una imagen *on*.

Figura 9. Código del botón iniciar grabación en memoria

```
    public void onClick( View view ) {
        execSend();
    }
});


  btiniciar.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if(grabando==true){
                grabando=false;
                btiniciar.setText("iniciar");
                indicador.setImageResource(R.drawable.off);
            }
            else{
                grabando=true;
                btiniciar.setText("parar");
                indicador.setImageResource(R.drawable.on);
                }
        }
    });
```

Elaboración propia

Para almacenar los datos obtenidos en la variable "str1", se almacenan en una variable local llamada "contenido", luego esta es escrita mediante un *Output Stream Writer* en la memoria SD del celular. Un *flush* corta la comunicación y se cierra la función *Output Stream* mediante un *close()*.

Figura 10. Código de proceso de almacenamiento en memoria

```
public void grabar(View v) {
       String nomarchivo = et1.getText().toString();
       String contenido = str1;
       try {
           File tarjeta = Environment.getExternalStorageDirectory();
           File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
           OutputStreamWriter osw = new OutputStreamWriter(
                   new FileOutputStream(file));
           osw.write(contenido);
           osw.flush();
           osw.close();
           Toast.makeText(this, "Los datos fueron grabados correctamente",
                   Toast.LENGTH_SHORT).show();
           et1.setText("");

       } catch (IOException ioe) {
       }
   }
```

Elaboración propia

Dentro de la función *exec Recieve*, que recibe los datos si estos son diferentes a un valor nulo, se consulta si se encuentra en modo de grabación y en caso que sea así se acumula la variable anterior con la actual, mostrando el texto de color verde como indicador. En caso contrario, la variable *str1* se mantiene con los datos anteriores, cambiando solo el texto a color blanco.

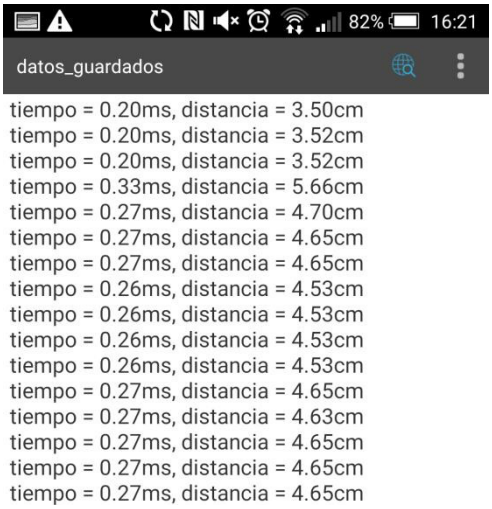
Figura 11. Código de cambio de color si está en modo grabación o no

```
protected void execRecieve( byte[] bytes ) {
    String str = bytesToString( bytes );
    if ( str != null ) {


        if(grabando==true)
        {   str1=str1+str;
            log.setTextColor(Color.GREEN);
        }
        else
        {
            log.setTextColor(Color.WHITE);
        }
        Log(str);
    }
}


 public void grabar(View v) {
       String nomarchivo = et1.getText().toString();
       String contenido = str1;
```

Elaboración propia

Para guardar los datos en memoria se elige un nombre alfanumérico y se presiona el botón *grabar*, almacenando automáticamente los datos en un texto plano.

Figura 12. Datos registrados



Elaboración propia

5. Conclusiones y recomendaciones

5.1. Conclusiones

i. Se creó una aplicación que permite registrar datos obtenidos de un sistema embebido en un dispositivo móvil por comunicación serial-USB.

ii. Se logró la utilización de un dispositivo móvil para el registro de datos, obteniéndose una ventaja de comunicación y al mismo tiempo como fuente de alimentación al sistema embebido.

iii. Se logró crear un programa en Java usando el entorno de Eclipse.

iv. El almacenamiento en la memoria SD de un dispositivo móvil ofrece una gran ventaja, porque no involucra la conexión a Internet lo que permite al usuario realizar un posterior análisis de la información recolectada.

5.2. Recomendaciones

i. A la aplicación se le podría implementar un visualizador gráfico que permita ver la evolución de una entrada analógica, en el tiempo que se realiza el muestreo.

ii. Se le podría agregar a la aplicación una ventana que permita ver el estado de las entradas digitales y analógicas de un sistema embebido.

Referencias

Almacenamiento de datos en un archivo de texto localizado en una tarjeta SD. (s.f.). Recuperado de http://www.tutorialesprogramacionya.com/javaya/androidya/ detalleconcepto.php?codigo=144&inicio=

Ohwada, K. (2014). *Android_UsbSerialTerminal1*. Recuperado de https://github.com/ ohwada/Android_UsbSerialTerminal1