

INSTITUTO TECNOLÓGICO DE CHILPANCINGO

Ingeniería en Sistemas Computacionales

**BuetoothScanner**

Taller de Cómputo Móvil

M. C. José Mario Martínez Castro

Alumno: Fernando Lucena Calixto

VERANO 2016

**Chilpancingo, Gro. Julio del 2016**

**BuetoothScanner**

[Bluetooth](https://en.wikipedia.org/wiki/Bluetooth) se ha convertido en una muy popular tecnología, especialmente en dispositivos móviles. Es una tecnología para descubrir y transferir datos entre dispositivos cercanos. Virtualmente cada dispositivo móvil moderno tiene capacidades Bluetooth éstos días. Si quieres crear una interfaz de aplicación con otro dispositivo con Bluetooth habilitado, desde teléfonos hasta altavoces, debes saber cómo usar la API de Bluetooth de Android.

En éste tutorial, crearemos una aplicación que es similar a la aplicación con Bluettoth integrado en la configuración de Android. Incluirá las siguientes características:

- habilitar Bluetooth en un dispositivo

- desplegar una lista de dispositivos asociados

- descubrir y listar dispositivos Bluetooth cercanos

Iremos sobre las bases para conectar y enviar datos a otro dispositivo Bluetooth. He creado un proyecto para iniciarnos, el cual puedes descargar desde [Github](https://github.com/fercho0?tab=overview&from=2016-07-09). La captura de pantalla de abajo muestra como se ve el proyecto de inicio. Si te estancas o encuentras problemas, entonces puedes observar el proyecto finalizado en [Githb](https://github.com/fercho0/BluetoothScanner).

[](https://github.com/fercho0/BluetoothScanner/blob/master/img/b1.png)

**Habilitar Bluetooth**

Antes de que podamos habilitar Bluetooth en un dispositivo Android, necesitamos requerir los permisos necesarios. Hacemos esto en el manifiesto de la aplicación. El permiso BLUETOOTH deja que nuestra aplicación se conecte, se desconecte, y transfiera datos con otro dispositivo Bluetooth. El permiso [BLUETOOTH\_ADMIN](https://developer.android.com/reference/android/Manifest.permission.html) deja que nuestra aplicación descubra nuevos dispositivos Bluetooth y cambia los ajustes sobre Bluetooth del dispositivo.

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.tutsplus.matt.bluetoothscanner" >

<uses-permission android:name="android.permission.BLUETOOTH" />

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

Usaremos el adaptador Bluetooth para conectarse mediante interfaz con Bluetooth. Instanciamos el adaptador en la clase ListActivity. Si el adaptador es null, esto significa que Bluetooth no es soportado por el dispositivo y la aplicación no funcionará en el dispositivo actual. Manejamos ésta situación mostrando un diálogo de alerta al usuario y saliendo de la aplicación.

@Override

protected void onCreate(Bundle savedInstanceState) {

BTAdapter = BluetoothAdapter.getDefaultAdapter();

// Phone does not support Bluetooth so let the user know and exit.

if (BTAdapter == null) {

new AlertDialog.Builder(this)

.setTitle("Not compatible")

.setMessage("Your phone does not support Bluetooth")

.setPositiveButton("Exit", new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int which) {

System.exit(0);

}

})

.setIcon(android.R.drawable.ic\_dialog\_alert)

.show();

}

}

Si Bluetooth está disponible en el dispositivo, necesitamos habilitarlo. Para habilitar Bluetooth, comenzamos un intent (intención) proporcionado a nosotros por el Android SDK, [BluetoothAdapter.ACTION\_REQUEST\_ENABLE](https://github.com/fercho0/BluetoothScanner/blob/master). Ésto presentará un diálogo al usuario, pidiéndole permiso para habilitar Bluetooth en el dispositivo. [REQUEST\_BLUETOOTH](https://github.com/fercho0/BluetoothScanner/blob/master) es una variable estática de tipo entero que declaramos para definir la petición de actividad.

public class ListActivity extends ActionBarActivity implements DeviceListFragment.OnFragmentInteractionListener {

public static int REQUEST\_BLUETOOTH = 1;

...

protected void onCreate(Bundle savedInstanceState) {

...

if (!BTAdapter.isEnabled()) {

Intent enable = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(enableBT, REQUEST\_BLUETOOTH);

}

}

}

**Obtener una Lista de Dispositivos Asociados**

En este paso, buscamos dispositivos Bluetooth y los desplegamos en una lista. En el contexto de un dispositivo móvil, un dispositivo Bluetoot puede ser:

- desconocido

- asociado

- conectado

Es importante saber la diferencia entre un dispositivo Bluetooth asociado y conectado. Dispositivos asociados se percatan de la existencia del otro y comparten una [clave de enlace](https://en.wikipedia.org/wiki/Bluetooth#Implementation_2), la cual puede ser usada para autenticar, resultando en una conexión. Los dispositivos son automáticamente asociados una vez que se establece una conexión encriptada.

Dispositivos conectados comparten un canal [RFCOMM](https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#Radio_frequency_communication_.28RFCOMM.29), permitiéndoles enviar y recibir datos. Un dispositivo puede tener muchos dispositivos asociados, pero solo puede ser conectado a un dispositivo a la vez.

Dispositivos Bluetooth son representados por el objeto BluetoothDevice. Una lista de dispositivos asociados pueden obtenerse al invocar el método getBondedDevices(), que retorna un conjunto de objetos BluetoothDevice. Invocamos el método getBondedDevices() en el método onCreate() de DeviceListFragment

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

Log.d("DEVICELIST", "Super called for DeviceListFragment onCreate\n");

deviceItemList = new ArrayList<DeviceItem>();

Set<BluetoothDevice> pairedDevices = bTAdapter.getBondedDevices();

}

Usamos los métodos getName() y getAddress() para obtener más información sobre los dispositivos Bluetooth. El método getName() retorna el identificador público del dispositivo mientras que el método getAddress() retorna la [dirección MAC](https://en.wikipedia.org/wiki/MAC_address) del dispositivo, un identificador único que identifica el dispositivo.

Ahora que tenemos una lista de dispositivos asociados, creamos un objeto DeviceItem para cada objeto BluetoothDevice. Posteriormente añadimos cada objeto DeviceItem a un arreglo llamado deviceItemList. Usaremos éste arreglo para desplegar la lista de dispositivos Bluetooth asociados en nuestra aplicación. El código para desplegar la lista de objetos DeviceItem ya está presente en el proyecto de inicio.

if (pairedDevices.size() > 0) {

for (BluetoothDevice device : pairedDevices) {

DeviceItem newDevice= new DeviceItem(device.getName(),device.getAddress(),"false");

deviceItemList.add(newDevice);

}

}

**Descubrir Dispositivos Bluetooth Cercanos**

El próximo paso es descubrir dispositivos con los que el dispositivo no está asociado todavía, dispositivos desconocidos, y agregarlos a la lista de dispositivos asociados. Hacemos ésto cuando el usuario pulsa el botón scan. El código para manejar ésto se localiza en DeviceListFragment.

Primero necesitamos hacer un BroadcastReceiver y sobre escribir el método onReceive(). El método onReceive() es invocado cuando un dispositivo Bluetooth es encontrado.

El método onReceive() toma un intent como su segundo argumento. Podemos checar con que clase de intent está transmitiendo invocando getAction(). Si la acción es BluetoothDevice.ACTION\_FOUND, entonces sabemos que hemos encontrado un dispositivo Bluetooth. Cuando ésto ocurre, creamos un objeto DeviceItem usando el nombre y la dirección MAC del dispositivo. Finalmente, agregamos el objeto DeviceItem al ArrayAdapter para desplegarlo en nuestra aplicación.

public class DeviceListFragment extends Fragment implements AbsListView.OnItemClickListener{

...

private final BroadcastReceiver bReciever = new BroadcastReceiver() {

public void onReceive(Context context, Intent intent) {

String action = intent.getAction();

if (BluetoothDevice.ACTION\_FOUND.equals(action)) {

BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA\_DEVICE);

// Create a new device item

DeviceItem newDevice = new DeviceItem(device.getName(), device.getAddress(), "false");

// Add it to our adapter

mAdapter.add(newDevice);

}

}

};

}

Cuando el botón scan está habilitado, simplemente necesitamos registrar el receptor que acabamos de hacer e invocar el método startDiscover(). Si el botón scan es deshabilidado, cancelamos el registro del receptor e invocamos cancelDiscovery(). Ten en mente que la acción de descubrir absorbe muchos recursos. Si tu aplicación se conecta con otro dispositivo Bluetooth, siempre deberías cancelar descubrir antes de conectar.

También removemos el objeto ArrayAdapter, mAdapter, cuando comienza la acción descubrir. Cuando comenzamos a escanear, no queremos incluir dispositivos anteriores que ya no pudieran estar en el rango del dispositivo.

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_deviceitem\_list, container, false);

ToggleButton scan = (ToggleButton) view.findViewById(R.id.scan);

...

scan.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION\_FOUND);

if (isChecked) {

mAdapter.clear();

getActivity().registerReceiver(bReciever, filter);

bTAdapter.startDiscovery();

} else {

getActivity().unregisterReceiver(bReciever);

bTAdapter.cancelDiscovery();

}

}

});

}

Eso es todo. Hemos finalizado nuestro scanner de Bluetooth.

**Conectar a un Dispositivo**

Conexiones Bluetooth funcionan como cualquier otra conexión. Hay un servidor y un cliente, que se comunican por medio de sockets (tomas) RFCOMM. En Android, las tomas RFCOMM se representan como un objeto BluetoothSocket. Afortunadamente para nosotros, la mayoría del código técnico para servidores es manejado por el Android SDK y disponible a través de la API de Bluetooth.

Conectarse como cliente es simple. Primero obtienes la toma RFCOMM del BluetoothDevice deseado al llamar a createRfcommSocketServiceRecord(), pasando un UUID, un valor de 128 bits que creas. El UUID es similar a un número de puerto.

Por ejemplo, asumamos que estás creando una aplicación de chat que usa Bluetooth para chatear con otros usuarios cercanos. Para encontrar otros usuarios con quienes chatear, querrías buscar otros dispositivos con la aplicación de chat instalada. Para hacer ésto, buscarías el UUID en la lista de servicios de los dispositivos cercanos. Usando un UUID para escuchar y aceptar las conexiones Bluetooth automáticamente agrega esa UUID a la lista de servicios del teléfono, o protocolo de descubrimiento de servicios.

Una vez que es creado el BluetoothSocket, llama a connect() en el BluetoothSocket. Ésto inicializará una conexión con BluetoothDevice a través de la toma RFCOMM. Una vez que nuestro dispositivo está conectado, podemos usar la toma para intercambiar datos con el dispositivo conectado. Hacer ésto es similar a cualquier implementación de servidor estándar.

Mantener una conexión Bluetooth es costoso así que necesitamos cerrar la toma cuando ya no lo necesitamos. Para cerrar la toma, llamamos a close() en el BluetoothSocket.

El siguiente fragmento de código muestra como conectar con un BluetoothDevice dado:

public class ConnectThread extends Thread{

private BluetoothSocket bTSocket;

public boolean connect(BluetoothDevice bTDevice, UUID mUUID) {

BluetoothSocket temp = null;

try {

temp = bTDevice.createRfcommSocketToServiceRecord(mUUID);

} catch (IOException e) {

Log.d("CONNECTTHREAD","Could not create RFCOMM socket:" + e.toString());

return false;

}

try {

bTSocket.connect();

} catch(IOException e) {

Log.d("CONNECTTHREAD","Could not connect: " + e.toString());

try {

bTSocket.close();

} catch(IOException close) {

Log.d("CONNECTTHREAD", "Could not close connection:" + e.toString());

return false;

}

}

return true;

}

public boolean cancel() {

try {

bTSocket.close();

} catch(IOException e) {

Log.d("CONNECTTHREAD","Could not close connection:" + e.toString());

return false;

}

return true;

}

}

Conectarse como servidor es un poco más difícil. Primero, desde tu BluetoothAdapter, debes obtener un BluetoothServerSocket, que será usado para escuchar una conexión. Esto se usa solamente para obtener la toma RFCOMM compartida de la conexión. Una vez que se establece la conexión, la toma del servidor ya no es necesaria y puede ser cerrada al llamar a close() en ella.

Instanciamos una toma del servidor al llamar listenUsingRfcommWithServiceRecord(String name, UUID mUUID). Éste método toma dos parámetros, un nombre de tipo String y un identificador único de tipo UUID. El parámetro nombre es el nombre que le damos al servicio cuando se agrega a la entrada SDP (Protocolo de Descubrimiento de Servicio) del teléfono. El identificador único debe coincidir con el UUID que el cliente está usando para tratar de conectar.

Entonces llamamos a accept() en el recientemente obtenido BluetoothServerSocket para esperar una conexión. Cuando la llamada a accept() retorna algo que no es null, lo asignamos a nuestro BluetoothSocket, que entonces podemos usar para intercambiar datos con el dispositivo conectado.

El siguiente fragmento de código muestra como aceptar una conexión como servidor:

public class ServerConnectThread extends Thread{

private BluetoothSocket bTSocket;

public ServerConnectThread() { }

public void acceptConnect(BluetoothAdapter bTAdapter, UUID mUUID) {

BluetoothServerSocket temp = null;

try {

temp = bTAdapter.listenUsingRfcommWithServiceRecord("Service\_Name", mUUID);

} catch(IOException e) {

Log.d("SERVERCONNECT", "Could not get a BluetoothServerSocket:" + e.toString());

}

while(true) {

try {

bTSocket = temp.accept();

} catch (IOException e) {

Log.d("SERVERCONNECT", "Could not accept an incoming connection.");

break;

}

if (bTSocket != null) {

try {

temp.close();

} catch (IOException e) {

Log.d("SERVERCONNECT", "Could not close ServerSocket:" + e.toString());

}

break;

}

}

}

public void closeConnect() {

try {

bTSocket.close();

} catch(IOException e) {

Log.d("SERVERCONNECT", "Could not close connection:" + e.toString());

}

}

}

Leer y escribir a la conexión se hace usando streams, InputStream y OutputStream. Podemos obtener una referencia a éstos streams llamando a getInputStream() y a getOutputStream() en el BluetoothSocket. Para leer desde y escribir a éstos streams, llamamos a read() y write() respectivamente.

El siguiente fragmento de código muestra cómo hacer esto para una sola variable de tipo entero:

public class ManageConnectThread extends Thread {

public ManageConnectThread() { }

public void sendData(BluetoothSocket socket, int data) throws IOException{

ByteArrayOutputStream output = new ByteArrayOutputStream(4);

output.write(data);

OutputStream outputStream = socket.getOutputStream();

outputStream.write(output.toByteArray());

}

public int receiveData(BluetoothSocket socket) throws IOException{

byte[] buffer = new byte[4];

ByteArrayInputStream input = new ByteArrayInputStream(buffer);

InputStream inputStream = socket.getInputStream();

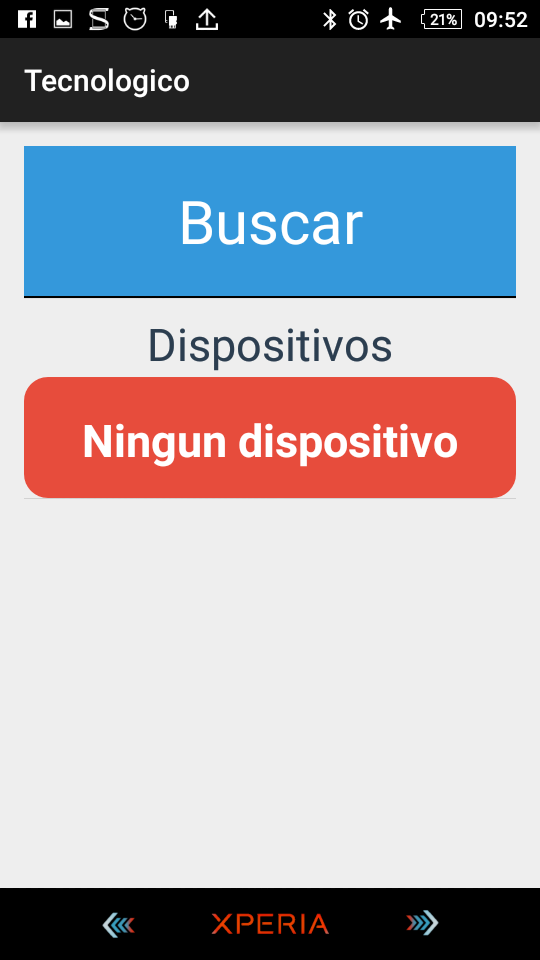
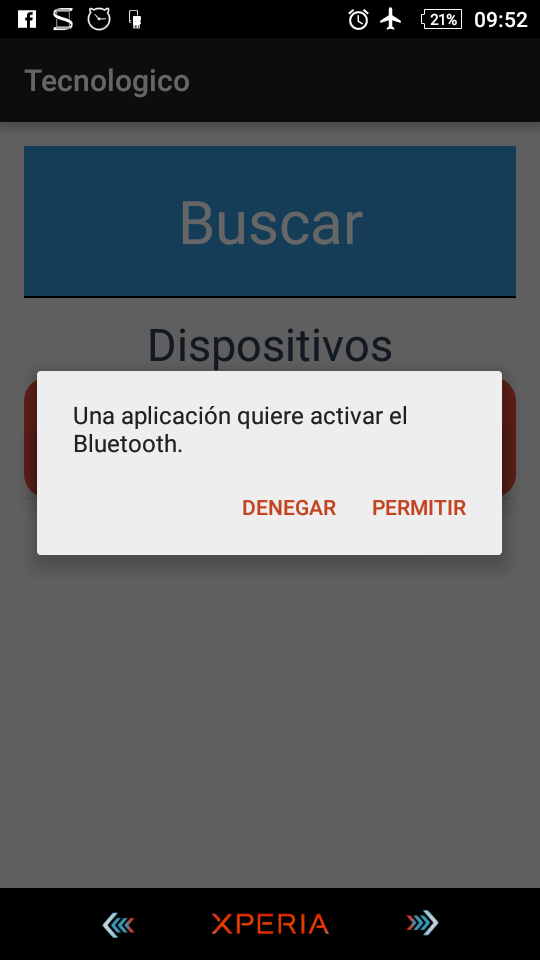
inputStream.read(buffer);

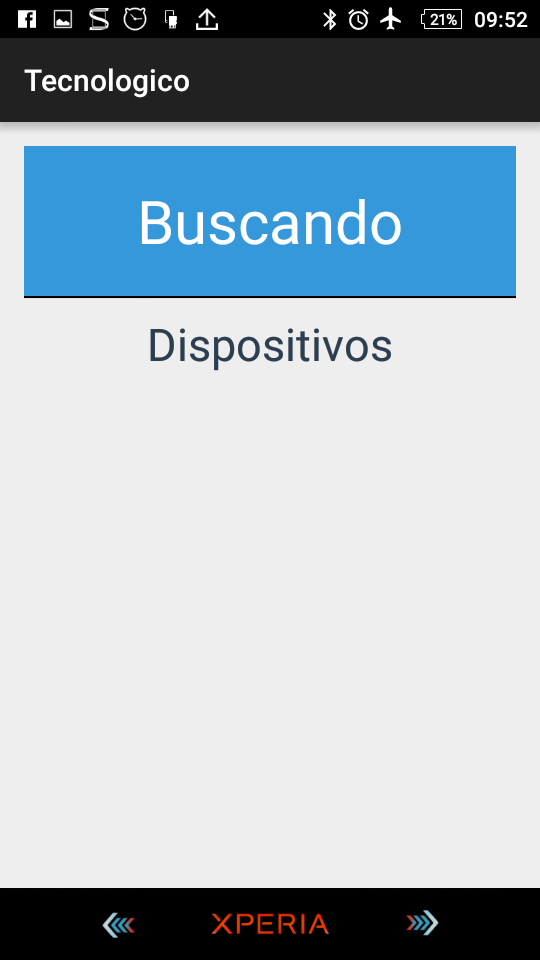
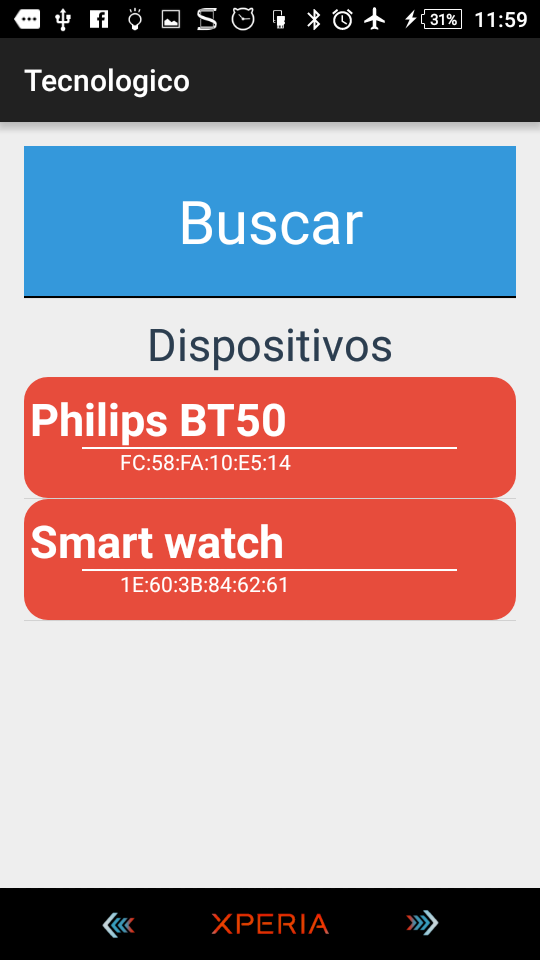
return input.read();

}

}

**capturas finales**

[](https://github.com/fercho0/BluetoothScanner/blob/master/img/b1.png)[](https://github.com/fercho0/BluetoothScanner/blob/master/img/b2.png)

[](https://github.com/fercho0/BluetoothScanner/blob/master/img/b3.png)[](https://github.com/fercho0/BluetoothScanner/blob/master/img/b4.png)