

ANDROID & ECLIPSE

Tutorial pour communiquer en Bluetooth avec le protocole SPP

Avant propos : Eclipse, le plugin Android SDK et un émulateur de tablette AVD sont installés.

1. Créer l'application

On opère comme d'habitude. Le projet et l'application sont nommés : "TestBluetoothSPP". L'activité principale se nomme "TestBluetoothActivity".

2. Description de l'application "TestBluetoothSPP"

Il s'agit d'établir une communication avec un module Bluetooth extérieur. Celui-ci est généralement connecté à un microcontrôleur qui désire communiquer avec la tablette ou le smartphone.

Le modèle utilisé est celui-ci :



Module hybride Bluetooth™ "FB755AS"

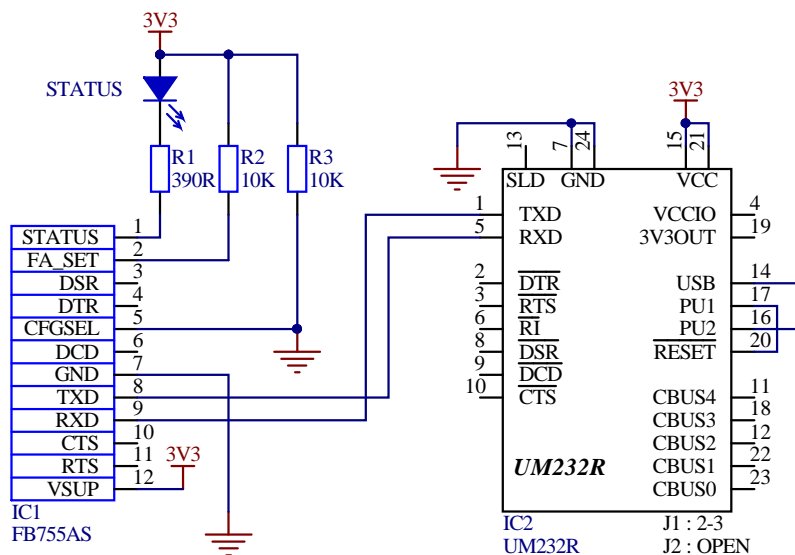
Le "FB755AS" de FIRMTECH est un module hybride DIL "OEM" subminiature Bluetooth™ Class 1 pré-qualifié faible consommation. Doté d'une antenne externe et d'une puissance d'émission de +12 dBm, il bénéficie d'une portée d'environ 100 mètres en terrain dégagé.

Extrêmement compact (20,5 x 27,7 x 12 mm - sans son antenne), performant et économique, le module hybride "FB755AS" est de part son format DIL (avec pas standard de 2,54 mm) associé à une "bonne" sensibilité (-83 dBm) et à une faible consommation, tout naturellement destiné à être intégré au sein d'applications embarquées les plus diverses. Le module est livré avec une petite antenne hélicoïdale (longueur 3 cm) déportée via un câble d'environ 10 cm relié sur un connecteur uFL.

Capable de gérer des communications Bluetooth™ conformément aux spécifications v2, le Firmware de base chargé dans le "FB755AS" lui permet de supporter le protocole de communication **SPP** (Serial Port Profile). Avec ce protocole, toutes les données arrivant sur le port série du "FB755AS" seront automatiquement transférées de façon transparente au périphérique connecté sur la liaison Bluetooth™. La communication étant bien évidemment bidirectionnelle.

Il est commercialisé par LEXTRONIC : <http://www.lextronic.fr/P20809-module-hybride-bluetooth-fb755as.html>

Pour les tests, ce module est connecté à un PC via une interface USB/RS232 comme le module UM232R de FTDI :



On ouvre alors un "HyperTerminal" paramétré comme il faut pour communiquer avec le module Bluetooth : vitesse, 8 bits, pas de parité, 1 bit stop et pas de protocole de poignée de main.

L'application est extrêmement simple :

- on établit avec la tablette Android la connexion avec le module Bluetooth
- on tape un caractère sur le clavier du PC
- il est transmis au module Bluetooth Firmtech
- celui-ci le transmet en Bluetooth suivant le protocole SPP
- la tablette le reçoit
- la tablette répond en renvoyant un paquet de 50 octets identiques à celui reçu.

En fait, le gros du codage de l'application sur la tablette consiste à obtenir une bonne convivialité pour établir la connexion Bluetooth :

- recherche des périphériques déjà enregistrés
- recherche des périphériques accessibles
- choix du périphérique
- et finalement : connexion

Pour ce faire, on utilisera un deuxième écran.

3. Projet "TestBluetoothSPP"

Le projet est constitué de 5 classes :

- "TestBluetoothSPPActivity" : activité principale. On reprend celle du projet de GrapheYT pour préparer l'application finale d'électrocardiographe (voir "Tutorial - Graphe YT de type roll.pdf"). Elle sera modifiée ici pour inclure l'utilisation du module Bluetooth.
- "GrapheYT" : version réduite de celle du projet de Graphe YT. Elle est créée pour éviter les erreurs de compilation.
- "ThreadGrapheYT" : thread du "GrapheYT". C'est une copie de celle du projet de Graphe YT.
- "BtListActivity" : nouvelle activité utilisée avec son écran spécifique pour visualiser les listes des périphériques déjà appairés, ceux détectés dans le voisinage et demander la connexion.
- "BluetoothService" : classe de type "Service" comportant 2 threads :
 - Le premier chargé de gérer la mise en connexion Bluetooth de la tablette avec le périphérique Firmtech et de renvoyer des messages (Intent) à l'activité principale. Celle-ci indique l'évolution de la connexion dans son titre.
 - Le deuxième chargé de gérer la connexion Bluetooth, notamment les échanges avec l'activité principale à la réception et la transmission des données.

L'utilisation de "thread" pour ces tâches permet d'éviter un blocage de la tablette dû à une boucle d'attente trop longue (attente de réception par exemple).

4. Objet « Graphe YT »

Cet objet n'a pas d'intérêt immédiat dans ce tutorial, il est là pour préparer l'application finale de réalisation d'un électrocardiographe.

Il comportera dans la version complète la définition de la surface graphique, les constantes et variables nécessaires et les méthodes pour l'initialiser et le manipuler.

On commence par créer la classe associée à l'objet « GrapheYT », héritée de la classe "SurfaceView" et implémentée de l'interface "SurfaceHolder.Callback" (méthodes de communication des réponses de retour). Pour l'instant elle ne comporte aucun traitement effectif, mais on y place les méthodes nécessaires pour éviter les erreurs :

```
package fr.couffignal;

import android.content.Context;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GrapheYT extends SurfaceView implements SurfaceHolder.Callback
{

    public GrapheYT(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
    {
        // TODO Auto-generated method stub
    }
}
```

```

@Override
public void surfaceCreated(SurfaceHolder holder)
{
    // TODO Auto-generated method stub
}

@Override
public void surfaceDestroyed(SurfaceHolder holder)
{
    // TODO Auto-generated method stub
}

@Override
protected void onDraw(Canvas canvas)
{
    // TODO Auto-generated method stub
    super.onDraw(canvas);
}
}

```

5. Ecran "main"

On reprend celui du graphe YT pour préparer l'application finale

L'écran est composé avec l'aide de l'éditeur graphique du fichier « res/layout/main.xml » :

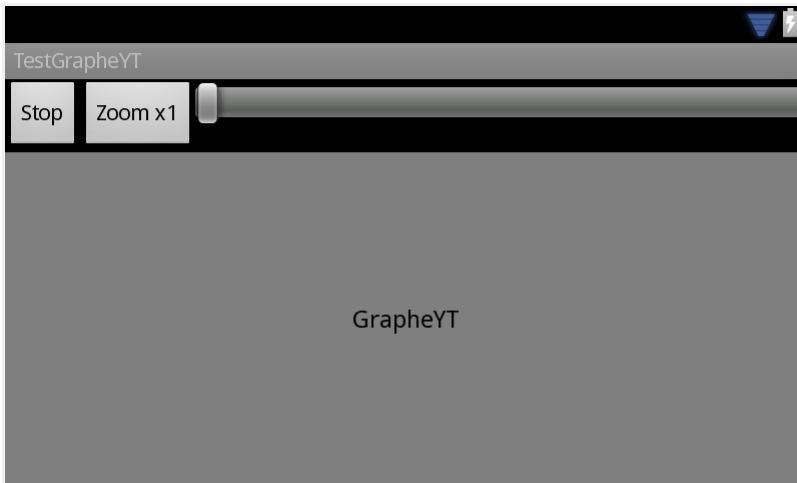
Fichier « main.xml » :

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <LinearLayout android:layout_height="wrap_content"
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent">
        <Button android:text="Stop"
            android:id="@+id/btnRunStopId"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <Button android:text="Zoom x1"
            android:id="@+id/btnZoomId"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <SeekBar android:id="@+id/curseurGrapheId"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:layout_width="match_parent">
        </SeekBar>
    </LinearLayout>
    <fr.couffignal.GrapheYT
        android:src="@android:drawable/screen_background_light"
        android:id="@+id/surfaceGrapheYT"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />
</LinearLayout>

```

On obtient l'écran suivant :



L'usage des boutons et du curseur ne sera pas développé ici (voir " Tutorial - Graphe YT de type roll.pdf").

6. « Thread » de dessin du graphe : "ThreadGrapheYT"

La classe "ThreadGrapheYT" n'a pas d'intérêt immédiat dans ce tutorial, elle est là pour préparer l'application finale de réalisation d'un électrocardiogramme. C'est une copie du projet de Graphe YT (voir " Tutorial - Graphe YT de type roll.pdf").

Sa fonction principale est le rafraîchissement du graphe déroulant. Ce traitement est réalisé par la méthode "onDraw" de la classe "GrapheYT" et est donc appelée régulièrement par le thread "ThreadGraphYT".

```

package fr.couffignal;

import android.graphics.Canvas;
import android.view.SurfaceHolder;

// Thread utilisé par "GrapheXY"
public class ThreadGrapheYT extends Thread
{
    private SurfaceHolder mySurfaceHolder;
    private GrapheYT myGrapheYT;
    private boolean runThreadGrapheYT; // Indicateur d'activité du "GrapheYT"

    // Constructeur de l'objet "MyThread"
    public ThreadGrapheYT(SurfaceHolder pSurfaceHolder, GrapheYT pGrapheYT)
    {
        mySurfaceHolder = pSurfaceHolder; // Objet "SurfaceHolder" du GrapheYT
        myGrapheYT = pGrapheYT; // Objet "GrapheYT"
    }

    // "Setter" de la variable "runThreadGrapheYT"
    public void setRunning(boolean run)
    {
        runThreadGrapheYT=run;
    }

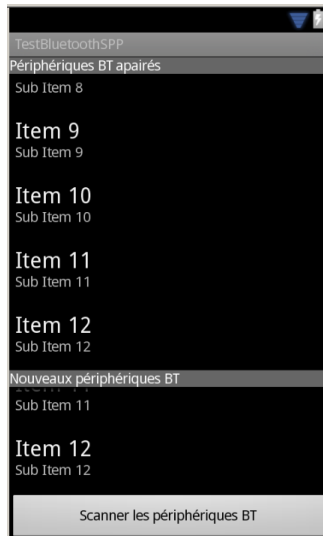
    // Méthode "run" du thread, appelée par le gestionnaire des tâches d'Android
    @Override
    public void run()
    {
        Canvas c;
        while (runThreadGrapheYT)
        { // Tant que le GrapheYT est actif
            c=null;
            try {
                c=mySurfaceHolder.lockCanvas(null); // Le canvas retourné permet
                // de dessiner sur la surface associée à "mySurfaceHolder",
                // soit "GrapheYT". Tous les pixels doivent être rafraichis
                synchronized (mySurfaceHolder){myGrapheYT.onDraw(c);} // Appel "onDraw"
            }
            finally {
                if (c!=null) {mySurfaceHolder.unlockCanvasAndPost(c);}
            }
        }
    }
}

```

Pour davantage de détails, se reporter au tutorial traitant du "Graphe YT".

7. Ecran "listes_bt"

Cet écran sera utilisé pour visualiser les listes des périphériques déjà appairés et ceux détectés dans le voisinage. On crée un nouveau fichier "xml" de nom "listes_bt" et on compose son contenu à l'aide de l'éditeur graphique :



"TextView" : pour le titre "périphériques BT appairés"

"ListView" : pour afficher la liste scrollable

"TextView" : pour le titre "Nouveaux périphériques BT"

"ListView" : pour afficher la liste scrollable

"Button" pour déclencher un scan des périphériques Bluetooth

Cela donne le fichier "listes_bt.xml" suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView android:text="Périphériques BT appairés"
    android:background="#666"
    android:textColor="#fff"
    android:id="@+id/textView1"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
  </TextView>
  <ListView android:id="@+id/paired_devices"
    android:layout_width="match_parent"
    android:stackFromBottom="true"
    android:layout_height="wrap_content"
    android:layout_weight="1">
  </ListView>
  <TextView android:text="Nouveaux périphériques BT"
    android:background="#666"
    android:textColor="#fff"
    android:id="@+id/textView2"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
  </TextView>
  <ListView android:id="@+id/new_devices"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:stackFromBottom="true"
    android:layout_weight="2">
  </ListView>
  <Button android:text="Scanner les périphériques BT"
    android:layout_height="wrap_content"
    android:id="@+id/scanBluetooth"
    android:layout_width="match_parent">
  </Button>
</LinearLayout>
```

Codage de la couleur :
RVB en hexa

Indique la liste la plus
"lourde" (la plus grande)

Indique la liste la moins
"lourde" (la plus petite)

Les "ListView" sont manipulées par les méthodes des objets "ArrayAdapter" comme on le verra dans les chapitres suivants.

8. Classe "BtListActivity"

Cette classe définit une nouvelle activité dont le but est de choisir le module Bluetooth "FIRMTECH" externe à connecter avec la tablette. Elle est ouverte par l'activité principale qui réagira aux choix effectués via l'interface "onClickListener".

Pour permettre le choix, l'activité liste les périphériques Bluetooth déjà associés et découvre sur demande les périphériques à portée. L'utilisateur choisit ensuite le périphérique dans les listes affichées.

La connexion effective est réalisée dans l'activité principale "TestBluetoothSPPActivity".

```

package fr.couffignal;

import java.util.Set;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class BtListActivity extends Activity implements OnClickListener
{
    // Return Intent extra
    public static String EXTRA_DEVICE_ADDRESS = "device_address";

    // Classe de l'adaptateur Bluetooth
    public static BluetoothAdapter monBluetooth;

    //Déclaration des instances des widgets de l'écran "listes_bt"
    Button btnScanBT;

    // Adaptateurs de tableaux pour manipuler les 2 "ListView"
    public ArrayAdapter mesPairedDevicesArrayAdapter;
    public ArrayAdapter mesNewDevicesArrayAdapter;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // Affichage de l'écran des "ListView"
        setContentView(R.layout.listes_bt);
        // Création de l'instance du bouton de l'écran "listes_bt"
        btnScanBT=(Button)findViewById(R.id.scanBluetooth);
        // Affecter par prudence le résultat CANCELED en cas d'arrêt prématuré
        setResult(Activity.RESULT_CANCELED);
        // "Ecouteur" du bouton "Scan Bluetooth"
        btnScanBT.setOnClickListener(this);
        // Initialisation des "ArrayAdapter". Un pour chaque type de liste de périphériques
        mesPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1);
        mesNewDevicesArrayAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1);
        // Création de l'instance "ListView" des périphériques appairés
        ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
        pairedListView.setAdapter(mesPairedDevicesArrayAdapter); // Association du "ArrayAdapter"
        pairedListView.setOnItemClickListener(mDeviceClickListener); // Association du "Listener"
        // Création de l'instance "ListView" des périphériques découverts
        ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
        newDevicesListView.setAdapter(mesNewDevicesArrayAdapter); // Association du "ArrayAdapter"
        newDevicesListView.setOnItemClickListener(mDeviceClickListener); // Association du "Listener"
    }
}

```

La classe "ArrayAdapter" comporte de nombreuses méthodes pour manipuler les "ListView"


```

// Enregistrement du filtre "BroadcastReceiver" qd un périphérique est découvert
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(mReceiver, filter);
// Enregistrement du filtre "BroadcastReceiver" qd la découverte est terminée
filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(mReceiver, filter);
// Affectation de la classe de l'adaptateur "Bluetooth"
monBluetooth=BluetoothAdapter.getDefaultAdapter();
// Lecture des périphériques Bluetooth déjà associés
Set<BluetoothDevice> pairedDevices = monBluetooth.getBondedDevices();
// Ajouter les noms et adresses des périphériques Bluetooth associés ds le "ArrayAdapter"
if (pairedDevices.size() > 0)
{ // L'affichage est mis à jour de par l'association du "ArrayAdapter" avec "pairedListView"
for (BluetoothDevice device : pairedDevices)
{
mesPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
}
}
else
{ // Aucun périphérique associé : "ArrayAdapter" affecté avec "Aucun peripherique associé"
mesPairedDevicesArrayAdapter.add("Aucun peripherique associé");
}
}

/*
 * Clic sur le bouton "Scan Bluetooth"
 */
@Override
public void onClick(View v)
{
doDiscovery(); // Changer le titre et démarrer la découverte des périphériques BT
};

// Méthode "on-click" appelée par un clic sur un item de toutes les "ListView"
public onItemClick(AdapterView<?> av, View v, int arg2, long arg3)
{
// Arrêt de la découverte car cette fonction est gourmande en énergie
if (monBluetooth.isDiscovering()) monBluetooth.cancelDiscovery();
// Lecture du nom et de l'adresse MAC dans les 17 derniers caractères du TextView v
String info = ((TextView) v).getText().toString(); // Nom et adresse MAC
String address = info.substring(info.length() - 17); // Adresse MAC dans les 17
// derniers caractères du TextView v

// Création de l'Intent de résultat et inclusion de l'adresse MAC
Intent intent = new Intent();
intent.putExtra(EXTRA_DEVICE_ADDRESS, address); // Avec l'identifiant utilisé ds
// l'activité principale ("onActivityResult")
// Affectation du résultat (OK) et arrêt de l'activité
setResult(Activity.RESULT_OK, intent); // Utilisé ds "onActivityResult" de l'activité princi.
finish();
}
};

/**
 * Démarrer la découverte des périphériques Bluetooth
 */
private void doDiscovery()
{
// Modifier le titre pour indiquer la phase "scanning"
setTitle("Scan Bluetooth ...");
// Stopper la découverte si elle est déjà en cours
if (monBluetooth.isDiscovering()) monBluetooth.cancelDiscovery();
// Démarrer la découverte des périphériques Bluetooth
monBluetooth.startDiscovery();
}

```

Filtres du "BroadcastReceiver" utilisés par la classe "monBluetooth"

Affectation de "mesPairedDevicesArrayAdapter" avec les périphériques Bluetooth déjà associés

Arg. "v" de type "TextView" ici contenant l'item choisi

La connexion effective avec le périphérique Bluetooth externe est demandée dans l'activité principale, dans la méthode "onActivityResult"

Méthode de découverte des périphériques Bluetooth

```
// Méthode "onReceive" du "BroadcastReceiver"
// Appelée en cas de message correspondant aux filtres "BluetoothDevice.ACTION_FOUND"
// ou "BluetoothAdapter.ACTION_DISCOVERY_FINISHED"
private final BroadcastReceiver mReceiver = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction(); // Lecture du message reçu depuis l'adaptateur BT
        // Identification du message
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            // Quand un nouveau périphérique est découvert :
            // Affectation du "BluetoothDevice" depuis l'Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // Non pris en compte s'il est déjà associé car il est déjà ds la liste
            if (device.getBondState() != BluetoothDevice.BOND_BONDED)
            {
                mesNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
        }
        // Nouveau titre à la fin de la découverte
        else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
        {
            setTitle("Sélectionner le périphérique");
            if (mesNewDevicesArrayAdapter.getCount() == 0)
            {
                // Aucun périphérique découvert
                mesNewDevicesArrayAdapter.add("Pas de périphérique trouvé");
            }
        }
    }
};

@Override
protected void onDestroy()
{
    super.onDestroy();
    // Arrêt de la découverte
    if (monBluetooth!=null) monBluetooth.cancelDiscovery();
    // Elimination du "BroadcastReceiver"
    this.unregisterReceiver(mReceiver);
}
}
```

Méthode "onReceive" du "BroadcastReceiver" appelée à chaque nouveau périphérique découvert et à la fin de la découverte

Affectation de "mesNewDevicesArrayAdapter" avec le périphérique Bluetooth découvert

Traitement à la fin de la découverte

Traitement à l'arrêt de l'activité

9. Code de la classe de services "BluetoothService"

Cette classe active 2 thread qui assurent les services suivants :

- "ConnectThread" est chargé de gérer la mise en connexion Bluetooth de la tablette avec le périphérique Firmtech et de renvoyer des messages (Intent) à l'activité principale. Celle-ci indique l'évolution de la connexion dans son titre.
- "ConnectedThread" est chargé de gérer la connexion Bluetooth, notamment les échanges avec l'activité principale à la réception et la transmission des données.


```

package fr.couffignal;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;

public class BluetoothService extends Service
{
    // UUID pour protocole SPP
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    // Classe de l'adaptateur Bluetooth
    private final BluetoothAdapter mAdapter; // Gestion du module Bluetooth de l'appareil
    // Handlers
    private Handler mHandler; // Handler de communication avec "UI Activity"
    private ConnectThread mConnectThread; // Thread de gestion de la mise en connexion
    private ConnectedThread mConnectedThread; // Thread de gestion de la connexion
    // Attribut de l'état actuel de la connexion
    private int mState;
    // Valeurs possibles de l'attribut "mState"
    public static final int STATE_NONE = 0; // Aucun traitement en cours
    public static final int STATE_LISTEN = 1; // Ecoute d'une connexion entrante (non utilisé)
    public static final int STATE_CONNECTING = 2; // Etablissement d'une connexion sortante
    public static final int STATE_CONNECTED = 3; // La connexion est établie

    /**
     * Constructeur. Préparation d'une nouvelle connexion Bluetooth
     * @param context Le contexte de l'activité UI
     * @param handler Le "Handler" utilisé pour transmettre des messages de retour vers
     * l'activité principale UI
     */
    public BluetoothService(Context context, Handler handler)
    {
        mAdapter=BluetoothAdapter.getDefaultAdapter(); // Instanciation de la classe locale "mAdapter"
        mState =STATE_NONE; // Aucun traitement en cours
        mHandler=handler; // Instanciation du Handler local
    }

    /**
     * "Setter" de l'état courant de la connexion
     * @param state Etat de la connexion
     */
    private synchronized void setState(int state)
    {
        mState = state;
        // Transmission de l'état vers l'activité principale UI
        mHandler.obtainMessage(TestBluetoothSPPActivity.MESSAGE_STATE_CHANGE,
            state, -1).sendToTarget();
    }

    /**
     * "Getter" qui renvoie l'état actuel de la connexion */
    public synchronized int getState()
    {
        return mState;
    }
}

```

UUID est l'abréviation de l'expression anglaise *Universally Unique Identifier*. Ce code est normalement utilisé pour crypter les communications. La valeur ci-dessous est identifiée par les 2 terminaux pour établir une communication série de protocole SPP (Serial Port Profil).

L'instanciation de l'objet "BluetoothService" est réalisée dans l'activité principale

Voir § 11

```

/**
 * Démarrer le "ConnectThread" pour établir une connexion avec le périphérique BT
 * @param device Le "BluetoothDevice" avec lequel on doit se connecter
 */
public synchronized void connect(BluetoothDevice device)
{
    // Arrêter d'abord tous les thread de connexion en cours
    if (mState == STATE_CONNECTING)
    {
        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    }
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    // Démarrer le thread de connexion avec le "BluetoothDevice" donné en paramètre
    mConnectThread = new ConnectThread(device); // Instanciation du thread
    mConnectThread.start(); // Démarrer le thread
    setState(STATE_CONNECTING); // Nouvel état de la connexion transmis à l'UI
}

/**
 * Démarrer le "ConnectedThread" pour maintenir une connexion avec le périphérique BT
 * @param socket Le "BluetoothSocket" de la connexion en cours
 * @param device Le "BluetoothDevice" du périphérique connecté
 */
public synchronized void connected(BluetoothSocket socket, BluetoothDevice device)
{
    // Arrêt du thread "mConnectThread" de mise en connexion
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    // Arrêt de l'éventuel thread de gestion d'une connexion établie
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    // Instanciation du thread de maintien de la connexion
    mConnectedThread = new ConnectedThread(socket);
    // Démarrer ce thread
    mConnectedThread.start();
    // Préparation du message qui sera affecté du nom du périphérique Bluetooth
    Message msg = mHandler.obtainMessage(TestBluetoothSPPActivity.MESSAGE_DEVICE_NAME);
    // Instanciation et affectation d'un "Bundle" avec ce nom
    Bundle bundle = new Bundle();
    bundle.putString(TestBluetoothSPPActivity.DEVICE_NAME, device.getName());
    // Renvoyer le nom du périphérique vers l'activité principale
    msg.setData(bundle);
    mHandler.sendMessage(msg);
    // Nouvel état de la connexion transmis à l'UI
    setState(STATE_CONNECTED);
}

/**
 * Methode d'arrêt de tous les threads
 */
public synchronized void stop()
{
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    setState(STATE_NONE); // Nouvel état de la connexion
}

/**
 * Ecriture d'un tableau d'octets dans le ConnectedThread de manière asynchrone
 * Si la connexion en cours le permet : transmission des octets au protocole SPP
 * @param out Le tableau d'octets à écrire
 * @see ConnectedThread#write(byte[])
 */
public void write(byte[] out)
{
    // Création d'un objet "ConnectedThread" temporaire
    ConnectedThread r;
    // Synchronisation de la copie du ConnectedThread
    synchronized (this)
    {
        if (mState != STATE_CONNECTED) return;
        // Uniquement en mode connecté
        r = mConnectedThread; // Instanciation de l'objet temporaire
    }
}

```

Méthode appelée dans l'UI en réponse au choix du périphérique

Par prudence, pour éviter les exceptions

Méthode appelée dans la méthode run du "ConnectThread" quand la connexion est établie

Méthode appelée à la fermeture de l'application

Méthode appelée dans l'UI pour transmettre un tableau d'octets en Bluetooth

```

// Ecriture effective du tableau d'octets
r.write(out);
}

/**
 * Méthode appelée quand une connexion n'a pas pu être établie
 * pour en notifier l'activité principale UI
 */
private void connectionFailed()
{
    setState(STATE_LISTEN); // Nouvel état de la connexion
    // Préparation et renvoi d'un message de perte vers l'activité principale
    Message msg = mHandler.obtainMessage(TestBluetoothSPPActivity.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(TestBluetoothSPPActivity.TOAST, "Connexion Bluetooth impossible");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
}

/**
 * Méthode appelée quand une connexion est rompue
 * pour en notifier l'activité principale UI
 */
private void connectionLost()
{
    setState(STATE_LISTEN); // Nouvel état de la connexion
    // Préparation et renvoi d'un message de rupture vers l'activité principale
    Message msg = mHandler.obtainMessage(TestBluetoothSPPActivity.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(TestBluetoothSPPActivity.TOAST, "Connexion Bluetooth perdue");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
}

/**
 * Ce thread est actif jusqu'à la connexion effective avec le périphérique BT
 * ou après un délai pré-déterminé.
 */
private class ConnectThread extends Thread
{
    private final BluetoothSocket mmSocket; // Interface de gestion d'une liaison BT RFCOMM
    private final BluetoothDevice mmDevice; // Objet associé au périphérique BT externe

    // Constructeur du thread appelé ds la méthode "connect"
    // L'argument "device" est affecté avec le périphériques BT choisi
    public ConnectThread(BluetoothDevice device)
    {
        mmDevice=device;
        BluetoothSocket tmpSocket=null; // Classe temporaire de gestion des canaux
        // d'écriture/lecture Bluetooth

        // Instanciation de la classe "BluetoothSocket"
        try
        {
            // Création d'un socket RFCOM avec l'UUID donné en argument
            tmpSocket=device.createRfcommSocketToServiceRecord(MY_UUID);
        }
        catch (IOException e)
        {
        }
        // Affectation du socket définitif avec le socket temporaire
        mmSocket=tmpSocket;
    }
}

```

Méthode "write" du
"ConnectedThread"

Classe "ConnectThread"

Constructeur de
"ConnectThread"

```

// Méthode "run" du "ConnectThread"
// Appelée régulièrement par le gestionnaire de tâche jusqu'à l'établissement
// de la connexion
public void run()
{
    setName("ConnectThread");
    // Arrêt de la découverte car cette fonction est gourmande en énergie
    if (mAdapter.isDiscovering()) mAdapter.cancelDiscovery();
    // Etablir une connexion avec le "BluetoothSocket"
    try
    {
        // Cet appel est bloquant et ne revient qu'après l'établissement de la
        // connexion ou en cas d'exception
        mmSocket.connect(); // Connexion effective. Le code PIN sera demandé
    }
    catch (IOException e)
    { // En cas d'exceptions : connexion BT impossible
        connectionFailed();
        try
        {
            mmSocket.close(); // Fermeture du "socket"
        }
        catch (IOException e2)
        {
        }
        return;
    }
    // Annulation synchronisée du ConnectThread
    synchronized (BluetoothService.this)
    {
        mConnectThread = null;
    }
    // Connexion établie : démarrer le "ConnectedThread"
    connected(mmSocket, mmDevice);
}

// Méthode de fermeture de ce thread
public void cancel()
{
    try
    {
        mmSocket.close(); // Fermeture du "Socket"
    }
    catch (IOException e)
    {
    }
}

/**
 * Ce thread est actif tant que la connexion avec le périphérique BT est établie
 * Elle gère les transmissions entrantes et sortantes
 */
private class ConnectedThread extends Thread
{
    private final BluetoothSocket mmSocket; // Interface de gestion d'une liaison BT RFCOMM
    private final InputStream mmInStream; // Classe de gestion du flux entrant
    private final OutputStream mmOutStream; // Classe de gestion du flux sortant

    // Constructeur du thread appelé ds la méthode "connected"
    // L'argument "socket" est affecté pour la connexion en cours
    public ConnectedThread(BluetoothSocket socket)
    {
        mmSocket = socket;
        InputStream tmpIn = null; // Classes temporaires de gestion des flux
        OutputStream tmpOut = null;
        // Instanciation des classes temporaires de gestion des flux entrant et sortant
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        }
        catch (IOException e)
    }
}

```

Méthode "run" de
"ConnectThread"

Classe "ConnectThread"

Constructeur de
"ConnectedThread"

```

    }
    mmInStream = tmpIn; // Instanciation des classes définitives si pas d'exception
    mmOutStream = tmpOut;
}

// Méthode "run" du "ConnectedThread"
// Appelée régulièrement par le gestionnaire de tâche jusqu'à la rupture
// de la connexion
public void run()
{
    byte[] buffer = new byte[100]; // Buffer temporaire
    int nbBytes; // Nombre d'octets reçus
    // L'écoute vers le "InputStream" est maintenue tant que la connexion est établie
    while (true)
    {
        try {
            // Lecture des octets reçus depuis le "InputStream"
            nbBytes = mmInStream.read(buffer);
            // Transmission du nb d'octets et du pointeur du buffer ds l'activité principale
            // qui traite les octets reçus
            mHandler.obtainMessage(TestBluetoothSPPActivity.MESSAGE_READ, nbBytes, -1, buffer)
                .sendToTarget();
        }
        catch (IOException e)
        {
            connectionLost(); // Perte de connexion en cas d'exception
            break;
        }
    }
}

/**
 * Méthode "write" vers le "OutStream"
 * Appelée par "BluetoothService.write"
 * @param buffer Les octets à écrire
 */
public void write(byte[] buffer)
{
    try {
        // Affectation de "mmOutStream" avec les octets à transmettre
        mmOutStream.write(buffer); // Transmission Bluetooth effective
        // Share the sent message back to the UI Activity
        // Transmission du pointeur du buffer ds l'activité principale qui ...
        // n'en fait rien dans cette application !
        mHandler.obtainMessage(TestBluetoothSPPActivity.MESSAGE_WRITE, -1, -1, buffer)
            .sendToTarget();
    }
    catch (IOException e)
    {
    }
}

// Méthode de fermeture de ce thread
public void cancel()
{
    try {
        mmSocket.close(); // Fermeture du socket
    }
    catch (IOException e)
    {
    }
}

@Override
public IBinder onBind(Intent intent)
{
    // TODO Auto-generated method stub
    return null;
}
}

```

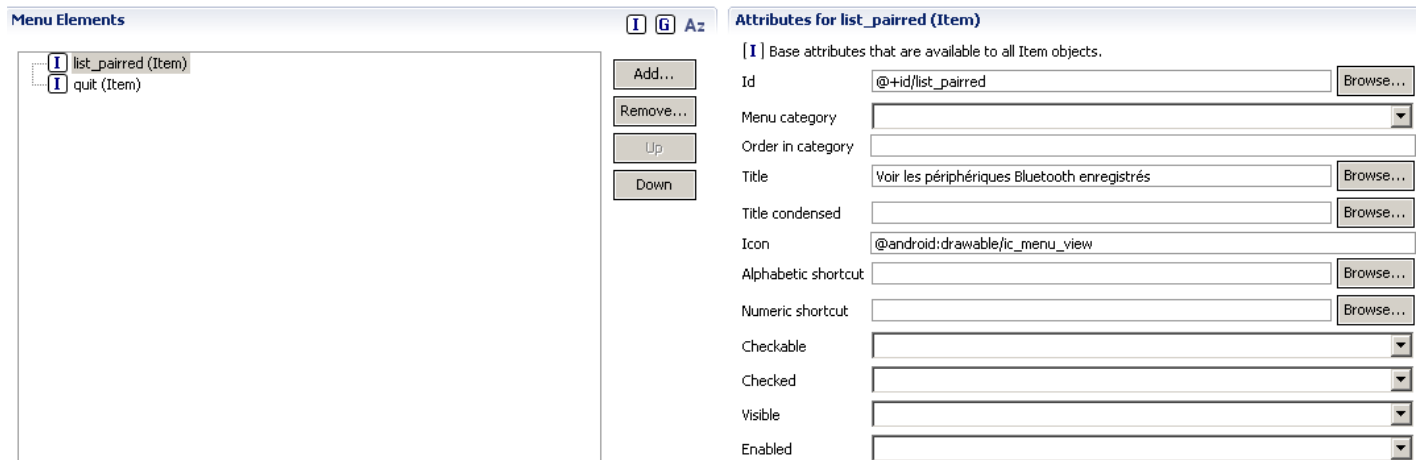
Méthode "run" de
"ConnectedThread"

Note : ne pas oublier de déclarer cette classe "service" dans "AndroidManifest.xml".

10. Menu de l'activité principale

Le menu est utilisé pour afficher les périphériques Bluetooth déjà associés ou quitter l'application. Pour cela, il faut créer un nouveau fichier "mon_menu.xml" et configurer ce menu :

 **Android Menu**



The screenshot shows the Eclipse IDE's 'Menu Elements' and 'Attributes for list_paired (Item)' panels. The 'Menu Elements' panel on the left shows a tree view with 'list_paired (Item)' and 'quit (Item)'. The 'Attributes for list_paired (Item)' panel on the right shows the configuration for the selected item. The 'Title' attribute is set to 'Voir les périphériques Bluetooth enregistrés' and the 'Icon' attribute is set to '@android:drawable/ic_menu_view'. Other attributes like 'Id', 'Menu category', 'Order in category', 'Title condensed', 'Alphabetic shortcut', 'Numeric shortcut', 'Checkable', 'Checked', 'Visible', and 'Enabled' are also visible with their respective values or default settings.

Ce qui donne le contenu suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/list_paired"
    android:title="Voir Les périphériques Bluetooth enregistrés"
    android:icon="@android:drawable/ic_menu_view">
  </item>
  <item android:id="@+id/quit"
    android:title="Quitter l'application"
    android:icon="@android:drawable/ic_menu_close_clear_cancel">
  </item>
</menu>
```

11. Code de la classe de l'activité principale "TestBluetoothSPPActivity"

```
package fr.couffignal;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

public class TestBluetoothSPPActivity extends Activity
{
  // Identificateurs des messages reçus du "Handler" de "BluetoothService"
  public static final int MESSAGE_STATE_CHANGE = 1;
  public static final int MESSAGE_READ = 2;
  public static final int MESSAGE_WRITE = 3;
  public static final int MESSAGE_DEVICE_NAME = 4;
  public static final int MESSAGE_TOAST = 5;

  // Noms clé reçus du "Handler" de "BluetoothService"
  public static final String DEVICE_NAME = "device_name";
  public static final String TOAST = "toast";

  // Codes de demande de l'Intent
  private static final int REQUEST_CONNECT_DEVICE = 1;
  private static final int REQUEST_ENABLE_BT = 2;
```

```

// Nom du périphérique Bluetooth connecté
private String mConnectedDeviceName = null;
// Classe de l'adaptateur Bluetooth local
private BluetoothAdapter mBluetoothAdapter = null;
// Classe du service "BluetoothService"
private BluetoothService mBluetoothService = null;

/** Méthode appelée qd l'activité est créée la première fois. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // Initialisation et affichage de l'écran principal
    setContentView(R.layout.main);
    // Recherche de l'adaptateur Bluetooth
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    // Test de présence de l'adaptateur
    if (mBluetoothAdapter == null)
    {
        // Absent : affichage d'un message spécifique et arrêt de l'application
        Toast.makeText(this, "Bluetooth non disponible", Toast.LENGTH_LONG).show();
        finish();
        return;
    }
}

@Override
protected void onStart()
{
    super.onStart();
    // L'adaptateur Bluetooth est-il déjà activé ?
    if (!mBluetoothAdapter.isEnabled())
    {
        // Non : demande d'activation de l'adaptateur via un Intent
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        // Une fenêtre de choix est demandée à l'utilisateur
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        // L'activité système répond par un appel de la méthode "onActivityResult"
    }
    else
    {
        // Oui : initialisation de "BluetoothService" pour gérer la connexion
        mBluetoothService = new BluetoothService(this, mHandler); // mHandler ci-dessous
    }
}

@Override
public void onDestroy()
{
    super.onDestroy();
    // Arrêt des services de "BluetoothService"
    if (mBluetoothService!=null) mBluetoothService.stop();
}

// Méthode non utilisée dans cette application
private void ensureDiscoverable()
{
    if (mBluetoothAdapter.getScanMode()!=BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)
    {
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivity(discoverableIntent);
    }
}

```

Arrêt prématuré de l'application si la tablette n'est pas équipée d'un module Bluetooth

Instanciation de l'Intent d'activation du module Bluetooth

Activation du module Bluetooth : cette activité « système » affiche une fenêtre permettant à l'utilisateur d'activer ou non le module Bluetooth

Module Bluetooth déjà actif : instanciation de l'objet « BluetoothService »


```

/*
 * Méthodes de gestion du menu
 * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
 */
// Constructeur
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.mon_menu, menu);
    return true;
}

// Méthodes de traitement du choix dans le menu
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    switch (item.getItemId())
    {
        case R.id.list_paired:
            // Choix : "Voir les périphériques Bluetooth enregistrés"
            Intent serverIntent = new Intent(this, BtListActivity.class);
            // Démarrer l'activité "BtListActivity" avec l'identifiant `
            // de réponse REQUEST_CONNECT_DEVICE
            startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
            return true;
        case R.id.quit: // Choix : "Quitter"
            finish(); // Fermeture de l'application
            return true;
    }
    return false;
}

// Méthode appelée à la fermeture d'une activité (activation BT ou « BtListActivity »)
// Elle identifie la réponse pour réaliser le traitement correspondant
public void onActivityResult(int requestCode, int resultCode, Intent data)
{
    switch (requestCode)
    {
        case REQUEST_ENABLE_BT:
            // L'activité système de demande d'activation de l'adaptateur Bluetooth a été
            // lancée avec cet identifiant
            if (resultCode == Activity.RESULT_OK)
            {
                // Le module Bluetooth est maintenant validé : démarrer "BluetoothService"
                // pour gérer la connexion et son maintien
                mBluetoothService = new BluetoothService(this, mHandler); // mHandler ci-dessous
                // Le Handler gère les messages de "BluetoothService" vers cette activité
            }
            else
            {
                // L'utilisateur a choisi de ne pas valider le module Bluetooth (ou erreur)
                Toast.makeText(this, "Le module Bluetooth n'est pas activé", Toast.LENGTH_SHORT).show();
                finish();
            }
        case REQUEST_CONNECT_DEVICE:
            // L'activité « BtListActivity » s'est terminée
            // L'activité "BtListActivity" a été lancée avec cet identifiant
            if (resultCode == Activity.RESULT_OK) // "BtListActivity" s'est terminée normalement
            {
                // Connexion au périphérique choisi
                // Lecture de l'adresse MAC du périphérique sélectionné
                String address = data.getExtras().getString(BtListActivity.EXTRA_DEVICE_ADDRESS);
                // Instanciation de l'objet "BluetoothDevice"
                BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
                // Connexion au périphérique
                mBluetoothService.connect(device); // Démarrer le "ConnectThread" pour établir la connexion
            }
            break;
    }
}
}

```

Le menu comporte 2 items :

- « list_paired » : afficher la liste des périphériques déjà enregistrés par l'activation du service « BtListActivity »
- « quit » : quitter l'application

Code d'identification de la réponse. Donnée au démarrage de l'activité.

L'activité système d'activation du module BT s'est terminée

L'utilisateur a choisi d'activer le module Bluetooth

L'activité « BtListActivity » s'est terminée

```

/*
 * Handler utilisé pour obtenir les messages du "BluetoothService"
 */
final Handler mHandler = new Handler()
{
    @Override
    public void handleMessage(Message msg)
    {
        switch (msg.what)
        {
            // Identifier le message
            case MESSAGE_STATE_CHANGE: // L'état du service a changé
                switch (msg.arg1)
                {
                    case BluetoothService.STATE_CONNECTING: // Connexion en cours
                        setTitle("Connection ...");
                        break;
                    case BluetoothService.STATE_CONNECTED: // Connexion établie
                        setTitle("Connecté à "+mConnectedDeviceName);
                        break;
                    case BluetoothService.STATE_LISTEN:
                    case BluetoothService.STATE_NONE: // Pas de connexion
                        setTitle("Non connecté.");
                        break;
                }
                break;
            case MESSAGE_WRITE: // Une écriture vers le périphérique a été déclenchée
                byte[] writeBuf = (byte[]) msg.obj; // Tableau d'octets transmis
                // On n'en fait rien ici !
                break;
            case MESSAGE_READ: // Des octets ont été reçus du périphérique Bluetooth
                byte[] readBuf = (byte[]) msg.obj; // Tableau contenant les octets reçus
                // Pour les tests : retransmission du tableau complet
                mBluetoothService.write(readBuf);
                break;
            case MESSAGE_DEVICE_NAME: // Réception du message comportant le nom du périphérique
                // Sauvegarde du nom du périphérique Bluetooth
                mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
                // Affichage temporaire de ce nom
                Toast.makeText(getApplicationContext(), "Connecté à "
                    + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
                break;
            case MESSAGE_TOAST: // Afficher un message "toast"
                Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
                    Toast.LENGTH_SHORT).show();
                break;
        }
    }
}

```

Instanciation du Handler avec l'objet « BluetoothService »

Premier argument de la fonction « handleMessage » du Handler appelée ds « BluetoothService »

Changer le titre pour voir l'évolution de la connexion

4° argument de la fonction « handleMessage » du Handler appelée ds « BluetoothService »

Attention : les octets ont déjà été transmis.

Traitement des octets recus

Les messages « Toast » ne peuvent être affichés que depuis une activité. Ils sont alors transmis via le Handler par les fonctions «connectionFailed » et « connectionLost » de « BluetoothService »