

# HoneyBee

Pour Particulier

**BILLOT – LUDWIG – LATTRECHE**

**Lycée La Briquerie**

**STIR2**

---

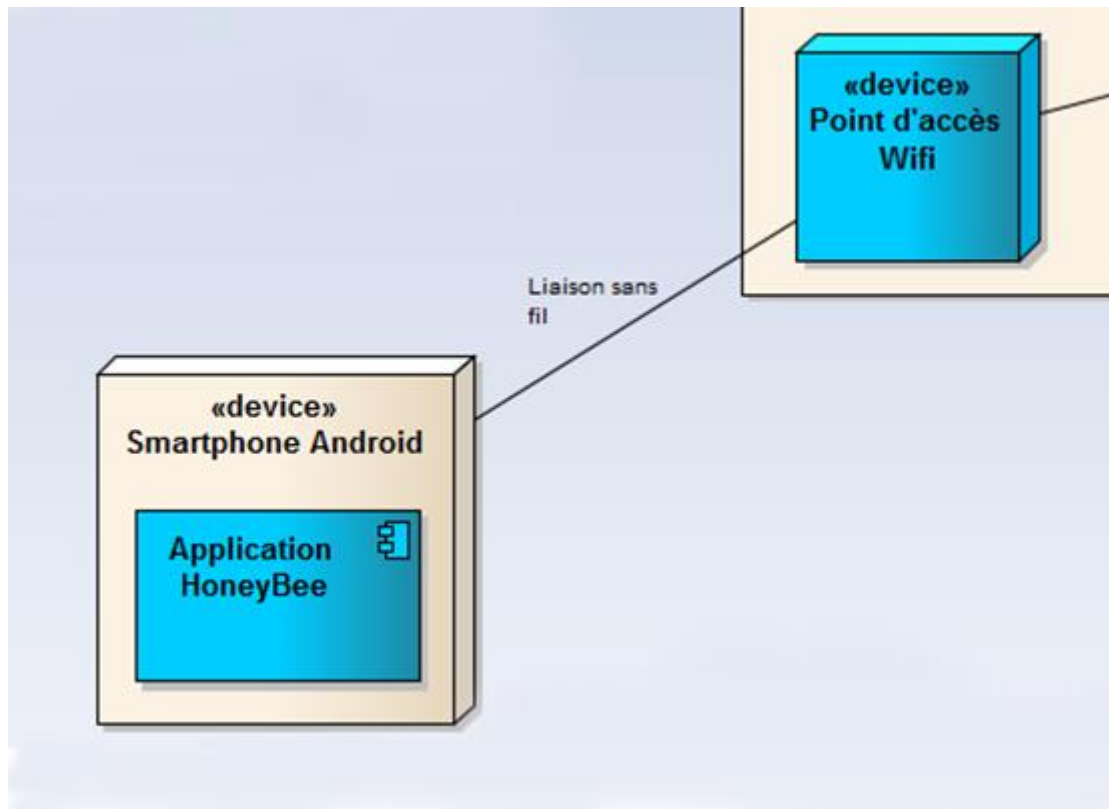
## Table des matières :

### 1 Contenu

Table des matières :	48
<b>2 Introduction</b>	<b>50</b>
<b>3 Présentation D'Android</b>	<b>51</b>
<b>4 Les Environnements de Développements</b>	<b>52</b>
4.1 Eclipse	52
4.2 Android Studio	53
<b>5 Protocole UDP</b>	<b>54</b>
5.1 Définition	54
5.2 Structure d'un datagramme UDP	54
5.3 Utilisation de l'UDP	55
<b>6 Test unitaire</b>	<b>56</b>
6.1 Connexion UDP entre deux PC	56
6.2 Compréhension de l'univers Android	58
6.2.1 Général	58
6.2.2 Android manifest .xml	59
6.2.3 Code java	60
6.2.4 Layout	61
6.3 Connexion UDP PC-ANDROID	61
6.4 Récupérer les données	63
<b>7 APPLICATION FINAL</b>	<b>65</b>
7.1 Le Service	65
7.2 Fonctionnement du service :	65
7.3 Utilisation du Service	66
7.4 Service UDP	66
7.4.1 Méthode onCreate()	66
7.4.2 Méthode onStartCommand()	67
7.5 WakeLock	69

7.6	Les notifications .....	69
7.7	Le bouton Retour.....	70
7.8	Graphisme final .....	71

## 2 Introduction



Ma partie du projet consiste à envoyer des données telles que le poids, la température de la ruche et la tension du calculateur Panda pour vérifier son niveau de batterie. La transmission des données se fait par wifi en utilisant le protocole UDP (**User Datagram Protocol**). L'utilisateur pourra alors voir les différentes données sur son Smartphone du jour, c'est-à-dire les données du matin dans les environs de 11h et 12h et aussi les données du soir dans les environs de 19h et 20h. L'utilisateur recevra aussi des mails en cas d'alerte importante de la ruche, par exemple si le poids est beaucoup trop faible ou si la batterie est faible.

### 3 Présentation D'Android



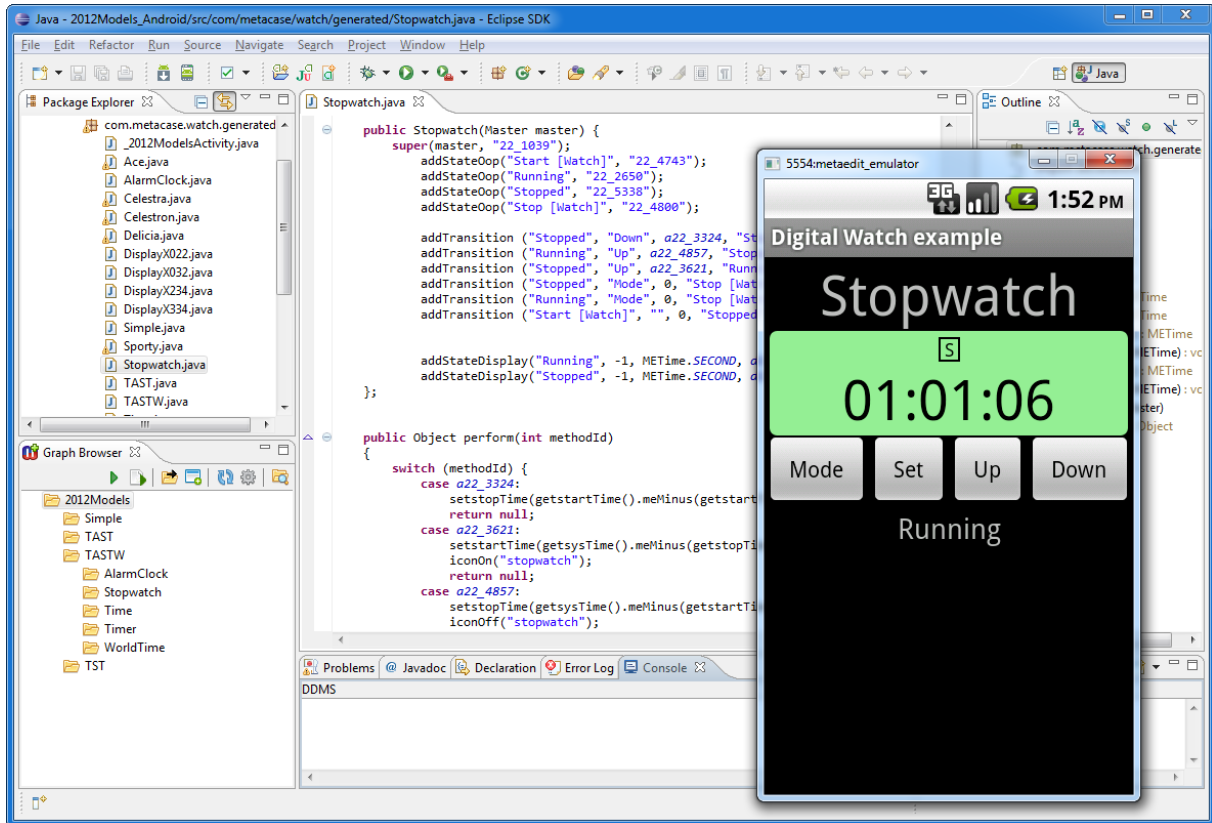
Android est un système d'exploitation mobile pour Smartphones, tablettes tactiles, PDA, montre connectée et terminaux mobiles. C'est un système open source utilisant le noyau Linux. Il a été lancé par une startup du même nom rachetée par Google en 2005. D'autres types d'appareils possédant ce système d'exploitation existent, par exemple des téléviseurs, des radioréveils, des autoradios et même des voitures.



\*Exemple d'un Smartphone fonctionnant sous Android, le Google Nexus 5.

## 4 Les Environnements de Développement

### 4.1 Eclipse



**Eclipse** est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la Fondation Eclipse visant à développer un environnement de production de logiciels libres qui soit extensible, universel et polyvalent, en s'appuyant principalement sur [Java](#).

Son objectif est de produire et fournir des outils pour la réalisation de logiciels, englobant les activités de programmation (notamment environnement de développement intégré et frameworks) mais aussi d'AGL recouvrant modélisation, conception, testing, gestion de configuration, reporting... Son EDI, partie intégrante du projet, vise notamment à supporter tout langage de programmation à l'instar de Microsoft Visual Studio.



Pour développer sur Android avec Eclipse il nous faut ajouter l'ADT ( Android Developer Tools) pour que le développement d'application puisse se dérouler

## 4.2 Android Studio



En 2014 est sorti l'environnement de développement créé par Google, intégrant directement tout ce qu'il faut pour programmer sur Android, c'est de ce fait que j'ai choisi d'utiliser ce logiciel de développement.

Android Studio représente la plateforme officielle, soutenue par Google, pour le développement d'applications Android. Il repose sur IntelliJ (Community Edition) de JetBrains et devrait permettre aux développeurs d'être plus rapides et plus productifs, puisqu'il intègre directement tout ce qu'il faut pour programmer sur Android.

## 5 Protocole UDP

### 5.1 Définition

Le **User Datagram Protocol (UDP)**, en français **protocole de datagramme utilisateur** est un des principaux protocoles de télécommunication utilisé par Internet. Il fait partie de la couche transport de la pile de protocole TCP/IP : dans l'adaptation approximative de cette dernière au modèle OSI, il appartiendrait à la couche 4, comme TCP.

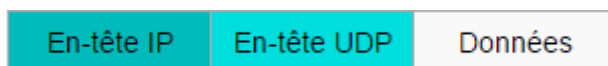
Le rôle de ce protocole est de permettre la transmission de données de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port. Contrairement au protocole TCP, il fonctionne sans négociation : il n'existe pas de procédure de connexion préalable à l'envoi des données. Donc UDP ne garantit pas la bonne livraison des datagrammes à destination, ni leur ordre d'arrivée. Il est également possible que des datagrammes soient reçus en plusieurs exemplaires.

L'intégrité des données est assurée par une somme de contrôle sur l'en-tête. L'utilisation de cette somme est cependant facultative en IPv4 mais obligatoire avec IPv6. Si un hôte n'a pas calculé la somme de contrôle d'un datagramme émis, la valeur de celle-ci est fixée à zéro. La somme de contrôle inclut les adresses IP source et destination.

La nature de UDP le rend utile pour transmettre rapidement de petites quantités de données, depuis un serveur vers de nombreux clients ou bien dans des cas où la perte d'un datagramme est moins gênante que l'attente de sa retransmission. Le DNS, la voix sur IP ou les jeux en ligne sont des utilisateurs typiques de ce protocole.

### 5.2 Structure d'un datagramme UDP

Le paquet UDP est encapsulé dans un paquet IP. Il comporte un en-tête suivi des données proprement dites à transporter.



L'en-tête d'un datagramme UDP est plus simple que celui de [TCP](#)



Port Source (16 bits)	Port Destination (16 bits)
Longueur (16 bits)	Somme de contrôle (16 bits)
Données (longueur variable)	

Il contient les quatre champs suivants :

### Port Source

Indique depuis quel port le paquet a été envoyé.

### Port de Destination

Indique à quel port le paquet doit être envoyé.

### Longueur

Indique la longueur totale (exprimée en octets) du segment UDP (en-tête et données). La longueur minimale est donc de 8 octets (taille de l'en-tête).

### Somme de contrôle

Celle-ci permet de s'assurer de l'intégrité du paquet reçu quand elle est différente de zéro. Elle est calculée sur l'ensemble de l'en-tête UDP et des données, mais aussi sur un pseudo en-tête (extrait de l'en-tête IP)

## 5.3 Utilisation de l'UDP

L'UDP est utilisé quand il est nécessaire soit de transmettre des données très rapidement et où la perte d'une partie de ces données n'a pas grande importance, soit de transmettre des petites quantités de données, là où la connexion TCP serait inutilement coûteuse en ressources. Par exemple, dans le cas de la transmission de la voix sur IP, la perte occasionnelle d'un paquet est tolérable dans la mesure où il existe des mécanismes de substitution des données manquantes, par contre la rapidité de transmission est un critère primordial pour la qualité d'écoute.

## 6 Test unitaire

### 6.1 Connexion UDP entre deux PC

Mon premier test unitaire constituait à réaliser une communication UDP entre deux PC pour comprendre comment fonctionne le protocole UDP.

Code du serveur :

```
private void b_envoyer_Click(object sender, EventArgs e)
{
    string message_entier;
    message_entier = t_poids.Text + " " + t_temperature.Text + " " + t_tension.Text;
    byte[] msg = Encoding.Default.GetBytes(message_entier);
    UdpClient serveur = new UdpClient();
    serveur.Send(msg, msg.Length, "127.0.0.1", 5053);
    serveur.Close();
}
```

Pour envoyer des trames UDP il faut que les données soient des bytes, pour ce faire je regroupe toute mes données en un string et l'insère dans un tableau de byte. Je crée ensuite un objet de type UdpClient qui permettra de créer une connexion UDP. La méthode send() permet d'envoyer le paquet à l'adresse choisit (ici 127.0.0.1) , mais la méthode prend différent paramètre comme le tableau de byte a envoyé , sa taille , l'adresse IP et le port sur lequel elle est émise.

Code du récepteur :

```
InitializeComponent();
_thecoute = new Thread(new ThreadStart(Ecouter));
_thecoute.Start();
```

On créer un Thread pour ne pas bloquer le processus principal

```
private void Ecouter()
{
    bool _continuer = true;
    UdpClient ecouteur = null;

    try
    {
        ecouteur = new UdpClient(5053);
    }
    catch
    {
        MessageBox.Show("impossible de se lier au port UDP 5053 , check reseau ");
    }

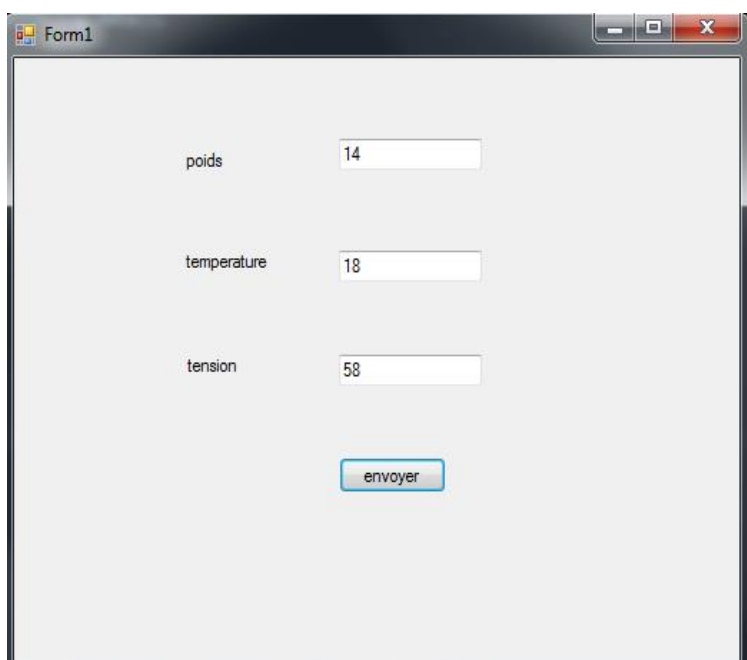
    while (_continuer)
    {
        IPEndPoint ip = null;
        byte[] data = ecouteur.Receive(ref ip);
        this.Invoke(new Action<string>(AjouterLog), Encoding.Default.GetString(data));
    }
}
```

Dans cette méthode écouter() , on crée un objet UdpClient qui écoute sur le port 5053 pour recevoir des données , dès qu'il entend quelque chose sur le port 5053, il le récupère.

```
string[] word = Regex.Split(data, " ");  
  
t_texte.AppendText(word[0]);  
t_tension.AppendText(word[2]);  
t_temp.AppendText(word[1]);
```

On affiche ensuite ces données sur l'interface principale.

Exemple :



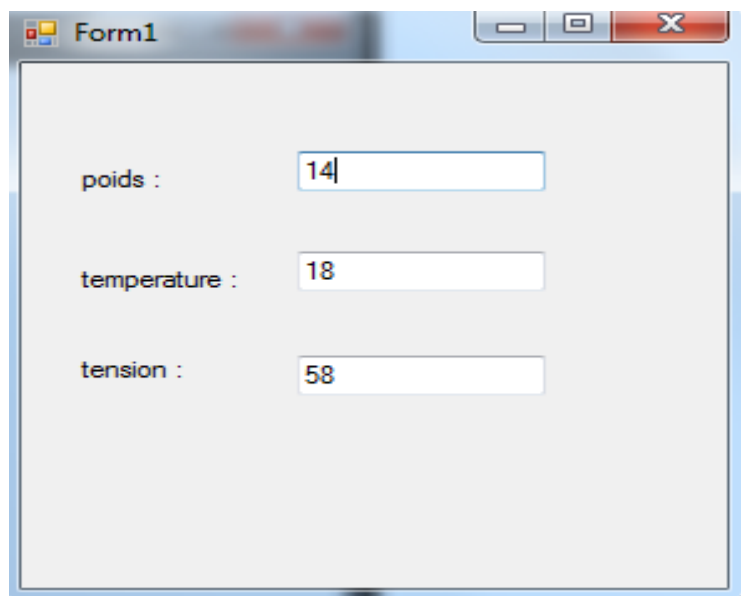
Form1

poids 14

temperature 18

tension 58

envoyer



Form1

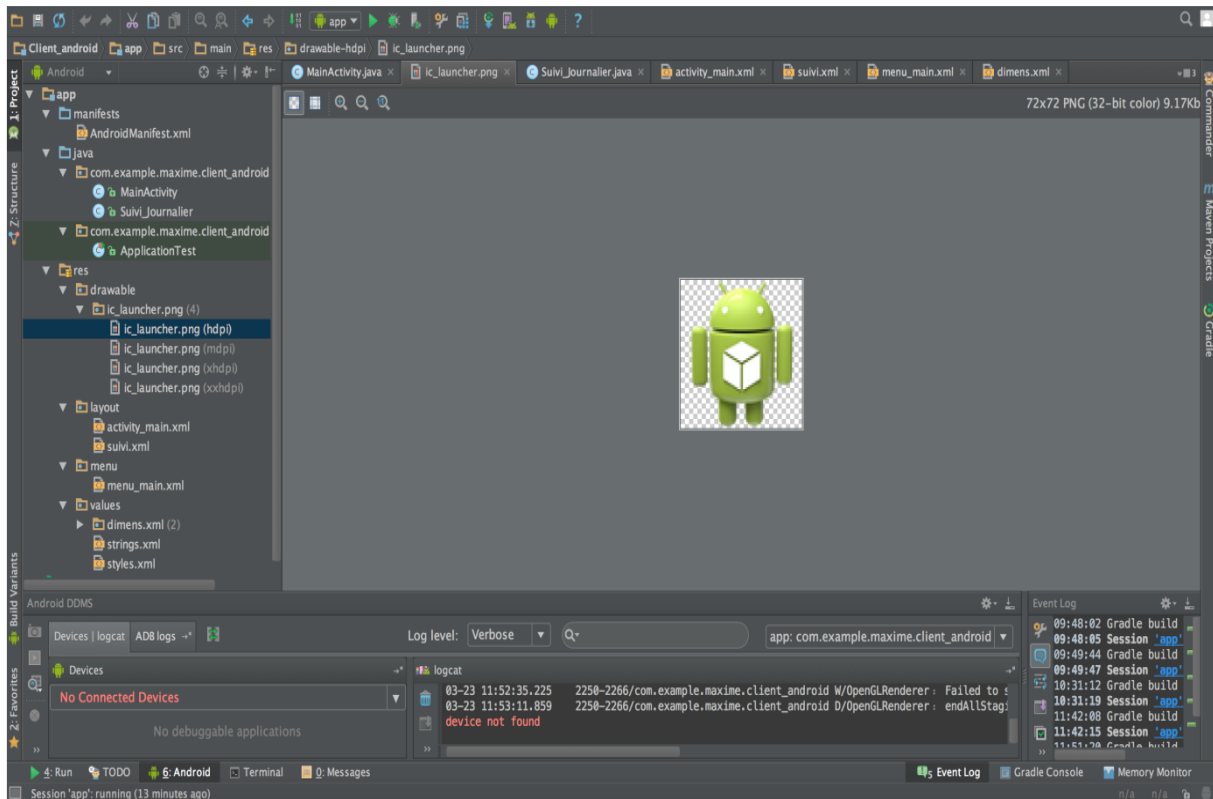
poids : 14

temperature : 18

tension : 58

## 6.2 Compréhension de l'univers Android

### 6.2.1 Général



#### Gestion des fichiers :

- Manifest : contient l'AndroidManifest.xml : définit le comportement de l'application au système android, ce fichier définit le nom, la version du SDK, les activités et les services
- MainActivity et Suivi\_Journalier sont mes principales activités, c'est ici que j'écris le code.
- Dossier res : ce dossier comporte les ressources de l'application (images, vidéo, styles)
- Drawable-hdpi : contient les images en hautes résolution
- Drawable-ldpi : contient les images en basse résolution
- Drawable-mdpi : contient les images en moyenne résolution
- Dossier layout : dossier décrivant les différentes interfaces
- Activity\_main.xml : le fichier principal de mon interface
- Dossier values : c'est un dossier contenant les valeurs de notre application, on peut par exemple y mettre des chaînes de caractères ou des tableaux.
- String.xml : contient les déclarations des chaînes de caractères.

## 6.2.2 Android manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.maxime.client_android" >

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="Client_android"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="Client_android" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Suivi_Journalier"
            android:label="@style/AppTheme">

        </activity>
    </application>
</manifest>
```

- Le package (identifiant) de notre application (ici com.example.maxime.client\_android) est défini dans l'attribut **package** de la balise manifeste.

- L'icône et le nom de notre application sont précisés par les attributs **icon** et **label** de la balise application ainsi que le thème utilisé par l'application (cf. chapitre Personnalisation et gestion d'événements - Personnalisation).

- La description de l'activité principale (balise **activity**) :

Le nom de la classe qui implémente Activity.

Le titre pour l'activité.

Des filtres sur l'activité :

- **MAIN** : indique qu'il s'agit de l'activité principale de l'application.

- **LAUNCHER** : indique que cette activité est présente dans le lanceur d'application.

### 6.2.3 Code java

Android a la particularité de programmer en JAVA.

Le langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

La société Sun a été ensuite rachetée en 2009 par la société Oracle qui détient et maintient désormais Java.

La particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tels que UNIX, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Pour cela, diverses plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en Java.



## 6.2.4 Layout

Les layouts permettent de créer des interfaces graphiques sur Android , il permet par exemple d'afficher des boutons , des textboxes ou encore des images .



## 6.3 Connexion UDP PC-ANDROID

Le test unitaire que j'ai effectué ensuite est la connexion entre Un pc et un Smartphone Android dans le même réseau wifi.

Je reprends alors le code de l'émetteur précédant

```
private void b_envoyer_Click(object sender, EventArgs e)
{
    string message_entier;
    message_entier = t_poids.Text + " " + t_temperature.Text + " " + t_tension.Text;
    byte[] msg = Encoding.Default.GetBytes(message_entier);
    UdpClient serveur = new UdpClient();
    serveur.Send(msg, msg.Length, "127.0.0.1", 5053);
    serveur.Close();
}
```

Le changement se fait dans le code Android

```
private class SocketClientThread extends Thread {

    DatagramPacket packet;
    DatagramSocket datagramSocket;
    Boolean continuer = true;
    String message2;
    @Override
    public void run()
    {
        try {
            datagramSocket = new DatagramSocket(5053);
        } catch (SocketException e1) {
            e1.printStackTrace();
        }
        byte [] buffer = new byte[1024];
        packet = new DatagramPacket(buffer,buffer.length);
        while (continuer) {

            try
            {
                datagramSocket.receive(packet);
                final String message = new String(packet.getData(),0,packet.getLength());
                message2=message +System.getProperty("line.separator");
                final String str [] = message.split(" ");

                FileOutputStream out = null;

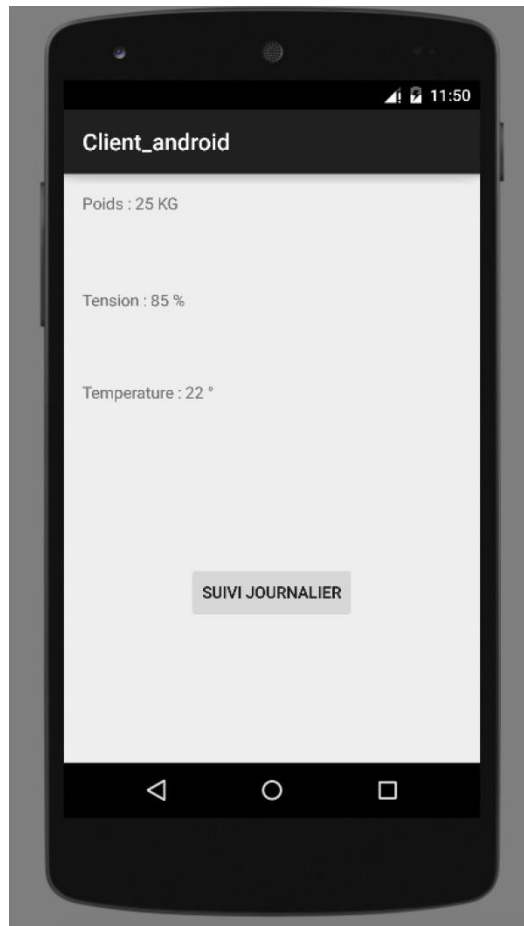
                try {
                    out = openFileOutput("test9.txt",MODE_APPEND);
                    out.write(message2.getBytes());
                }
                catch (FileNotFoundException e)
                {
                    e.printStackTrace();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }

                MainActivity.this.runOnUiThread(() -> {
                    poids.setText(str[2]);
                    tension.setText(str[3]);
                    temperature.setText(str[4]);
                });
            } catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

Je crée une classe publique Thread me permettant à n'importe quel moment de recevoir les informations du PC. J'enregistre dans un fichier texte la ligne qui vient d'être envoyée. Chaque donnée est séparée d'un espace pour pouvoir ensuite splitté au bon endroit et récupérer les données que je souhaite.

J'affiche ensuite les données sur l'interface du Smartphone :





## 6.4 Récupérer les données

La suite de mon application est de récupérer les valeurs des jours précédents, pour se faire je lis les informations dans mon fichier texte je vérifie que la date corresponde à la date que l'utilisateur a choisi, puis s'il voulait les données de 12h ou de 19h. L'utilisateur devra appuyer sur le bouton rechercher pour afficher les résultats

```
try {
    FileInputStream fstream = openFileInput("test9.txt");
    DataInputStream in = new DataInputStream(fstream);
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    while ((line=br.readLine())!=null)
    {
        String Txtdate=txtDate.getText().toString();
        String str [] = line.split(" ");

        if (groupe.getCheckedRadioButtonId() == R.id.choix1)
        {
            heure = "12:00";
        }
        else if (groupe.getCheckedRadioButtonId()== R.id.choix2)
        {
            heure ="19:00";
        }

        if (str[0].equals(Txtdate) && str[1].equals(heure))
        {
            poids.setText(str[2]);
            tension.setText(str[3]);
            temperature.setText(str[4]);
        }
    }
}
```



## 7 APPLICATION FINAL

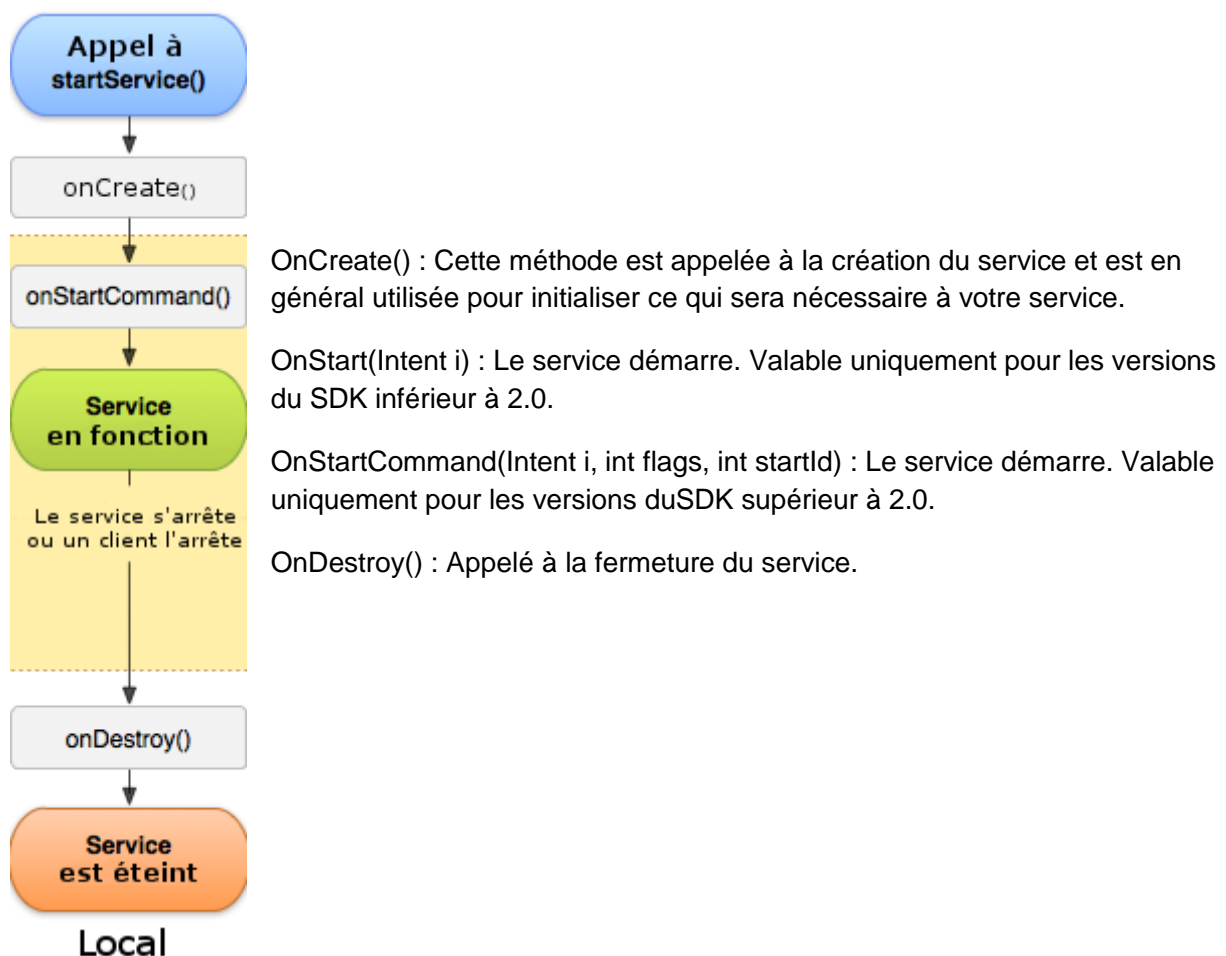
### 7.1 Le Service

Contrairement aux threads, les services sont conçus pour être utilisés sur une longue période de temps. En effet, les threads sont des éléments sommaires qui n'ont pas de lien particulier avec le système Android, alors que les services sont des composants et sont par conséquent intégrés dans Android au même titre que les activités. Ainsi, ils vivent au même rythme que l'application. Si l'application s'arrête, le service peut réagir en conséquence, alors qu'un thread, qui n'est pas un composant d'Android, ne sera pas mis au courant que l'application a été arrêtée si vous ne lui dites pas. Il ne sera par conséquent pas capable d'avoir un comportement approprié, c'est-à-dire la plupart du temps de s'arrêter.

Le service nous sera donc utile quand l'utilisateur aura l'application en fond de tâche, il permettra de recevoir les données à n'importe quel moment même si l'application est tuée, le service ne le sera pas.

J'ai donc créé une classe ServiceUDP :

### 7.2 Fonctionnement du service :



## 7.3 Utilisation du Service

Il faut d'abord le déclarer dans le Androidmanifest.xml

```
<service
    android:enabled="true"
    android:name="com.example.maxime.client_android.ServiceUDP"
/>
```

Je l'initialise dans le OnStart() de l'application principale , c'est-à-dire dès que l'application Honeybee se lance , le service sera lancé.

```
@Override
public void onStart()
{
    Intent intent = new Intent(MainActivity.this,ServiceUDP.class);
    startService(intent);
    super.onStart();
}
```

## 7.4 Service UDP

### 7.4.1 Méthode onCreate()

```
@Override
public void onCreate() {
    super.onCreate();
    try {
        datagramSocket = new DatagramSocket(5053);
    }
    catch (SocketException e)
    {

    }
    packet = new DatagramPacket(buffer,buffer.length);
}
```

Dans le onCreate() on initialise le DatagramSocket et le DatagramPacket.

### 7.4.2 Méthode onStartCommand()

```
@Override
public int onStartCommand (final Intent intent , int flags , int startID) {

    final String LOG_TAG = "MyClass";
    Log.w(LOG_TAG, "le startcommand s'est lancé");

    Thread UDPThread;
    UDPThread = new Thread((Runnable) () -> {
        while (continuer) {
            try
            {
                datagramSocket.receive(packet);
                Log.w(LOG_TAG, "le startcommand a reçu");
                final String message = new String(packet.getData(),0,packet.getLength());
                message2=message +System.getProperty("line.separator");
                Intent intent = new Intent();
                intent.setAction(MY_ACTION);
                intent.putExtra("DATAPASSED",message2);
                sendBroadcast(intent);
                Log.w(LOG_TAG, "le message est passé");

                try {
                    FileOutputStream out = openFileOutput("test12.txt",MODE_APPEND);
                    out.write(message2.getBytes());
                }
                catch (FileNotFoundException e)
                {
                    e.printStackTrace();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }

            } catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    });
}
```

On crée un Thread pour ne pas bloquer le processus principal, ensuite on récupère la trame envoyée avec `datagramSocket.receive(packet)`. On stocke ensuite cette donnée dans un string.

On crée ensuite un Intent pour faire passer la valeur du string « message2 » dans le MainActivity pour l'afficher à l'utilisateur les données reçues.

```
private class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context arg0 , Intent arg1)
    {
        Bundle bundle;
        bundle = arg1.getExtras();
        String Smessage = bundle.getString("DATAPASSED");
        final String str [] = Smessage.split("\n");
        poids.setText(str[2]);
        tension.setText(str[4]);
        temperature.setText(str[3]);
    }
}
```

On crée une classe qui hérite de `BroadcastReceiver` qui permet à l'activité principale de récupérer les valeurs du Service.

```
myReceiver = new MyReceiver();  
IntentFilter intentFilter = new IntentFilter();  
intentFilter.addAction(ServiceUDP.MY_ACTION);  
registerReceiver(myReceiver, intentFilter);
```

On crée une instance de la classe MyReceiver , puis avec L'intentFilter on exécute le code de l'objet myReceiver.

A la fin de onStartCommand() dans notre cas nous avons un return « **Service.StartSTicky** » mais il y a aussi des autres valeurs de retour possibles.

#### START\_NOT\_STICKY

Si le système tue le service, alors ce dernier ne sera pas recréé. Il faudra donc effectuer un nouvel appel à startService() pour relancer le service.

Ce mode vaut le coup dès qu'on veut faire un travail qui peut être interrompu si le système manque de mémoire et que vous pouvez le redémarrer explicitement par la suite pour recommencer le travail. Si vous voulez par exemple mettre en ligne des statistiques sur un serveur distant. Le processus qui lancera la mise en ligne peut se dérouler toutes les 30 minutes, mais, si le service est tué avant que la mise en ligne soit effectuée, ce n'est pas grave, on le refera dans 30 minutes.

#### START\_STICKY

Cette fois, si le système doit tuer le service, alors il sera recréé mais sans lui fournir le dernier Intent qui l'avait lancé. Ainsi, le paramètre intent vaudra null. Ce mode de fonctionnement est utile pour les services qui fonctionnent par intermittence, comme par exemple quand on joue de la musique.

#### START\_REDELIVER\_INTENT

Si le système tue le service, il sera recréé et dans onStartCommand() le paramètre intent sera identique au dernier intent qui a été fourni au service. START\_REDELIVER\_INTENT est indispensable si vous voulez être certain qu'un service effectuera un travail complètement.

## 7.5 WakeLock

```
PowerManager pm=(PowerManager) getSystemService(Context.POWER_SERVICE);
wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,"monwakeLock");
```

Le WakeLock permet au service de tourner même quand l'écran du smartphone est éteint, il y a plusieurs options possible :

FULL\_WAKE\_LOCK - garde l'écran et le clavier allumé et le processeur travaille

SCREEN\_BRIGHT\_WAKE\_LOCK – garde l'écran allumé et le processeur travaille

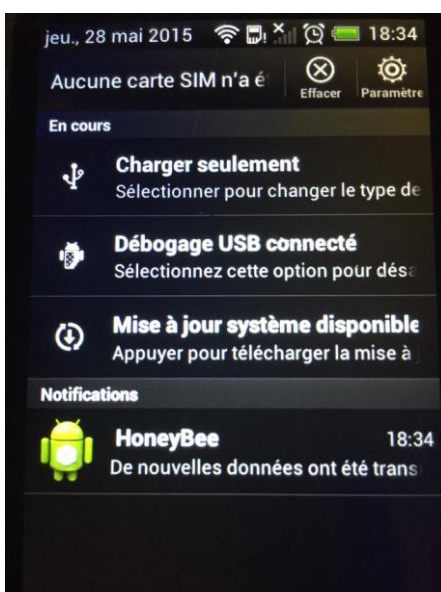
PARTIAL\_WAKE\_LOCK - seul le processeur travaille.

## 7.6 Les notifications

Une notification est une indication qui s'affiche sur la barre qui se situe en haut d'un téléphone Android. Cette notification sert à prévenir un utilisateur de certains événements, comme la réception d'un message par exemple.

```
NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
Notification notification = new Notification(R.drawable.ic_launcher,"HoneyBee",System.currentTimeMillis());
Intent notifintent = new Intent(ServiceUDP.this,MainActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(ServiceUDP.this,0,notifintent,0);
String bonjour = " HoneyBee";
String aurevoir = "De nouvelles données ont été transmises !";
notification.setLatestEventInfo(ServiceUDP.this,bonjour,aurevoir,pendingIntent);
notification.vibrate = new long[] {0,200,100,200,100,200};
notificationManager.notify(ID_NOTIFICATION, notification);
```

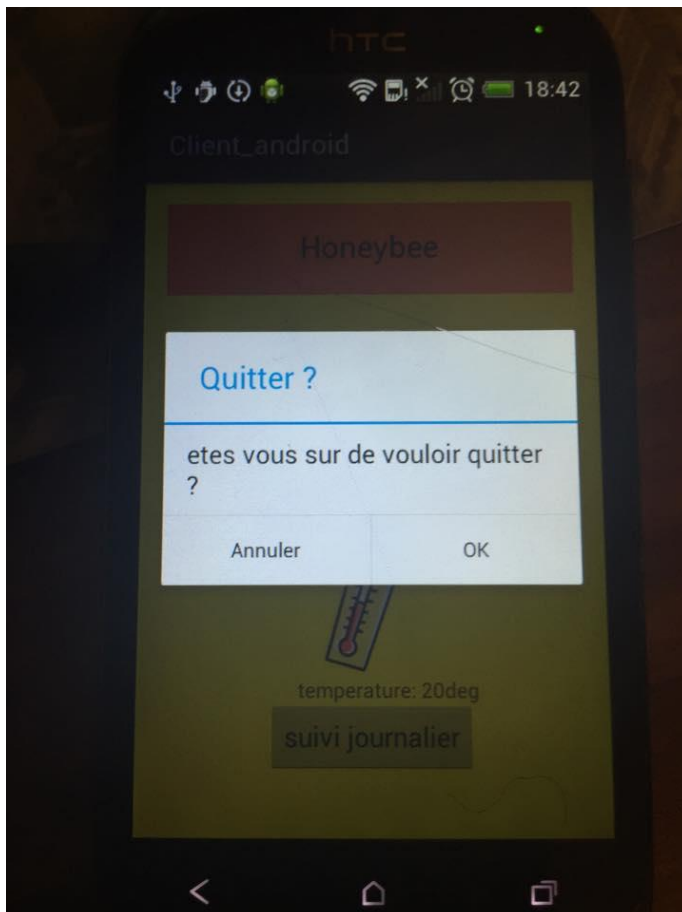
On crée une notification avec comme nom « honeybee » l'image ic.launcher , on crée aussi une vibration au moment de recevoir les données.



## 7.7 Le bouton Retour

```
@Override
public void onBackPressed(){
    new AlertDialog.Builder(this)
        .setTitle("Quitter ?")
        .setMessage("etes vous sur de vouloir quitter ? ")
        .setNegativeButton(android.R.string.no, null)
        .setPositiveButton(android.R.string.yes, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                MainActivity.super.onBackPressed();
                System.exit(0);
            }
        }).create().show();
}
```

Quand l'utilisateur appuiera sur la touche retour de son smartphone, l'application affichera une boîte de dialogue, où il pourra ensuite décider s'il veut complètement quitter l'application ou pas.





## 7.8 Graphisme final

