# Android TP 6 : Liaison Bluetooth

Dans ce TP, nous allons étudier comment implémenter une connexion Bluetooth sous Android.

Ce TP a été écrit à partir de la documentation d'ANDROID STUDIO. Un modèle de programmation du Bluetooth est proposé ici :

https://developer.android.com/guide/topics/connectivity/bluetooth

Le TP suit la démarche recommandée dans cette documentation, en n'en gardant que les points principaux.

On trouvera en annexe un rappel sur la programmation du widget « ListView »

## 1 Mise en place du projet

Le modèle de projet retenu sera celui du TP N°2, dans lequel nous avions créé 2 pages avec une navigation d'une page à l'autre dans un context de « Fragment ». Ceci pour prévoir un développement possible du projet de fin de formation (Figure 1).

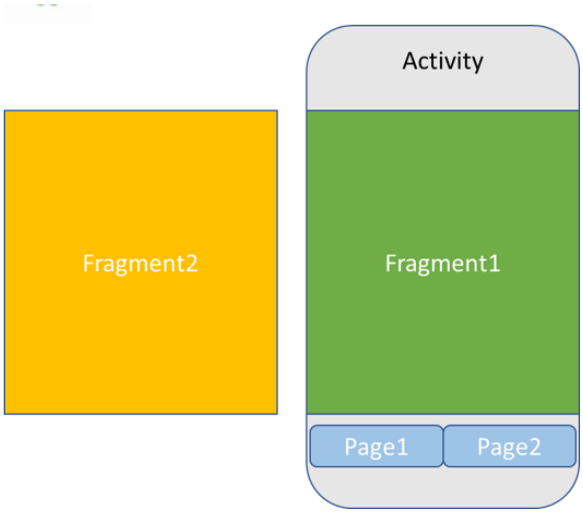



Figure 1 - Interface utilisateur

→ Faire une duplication du projet du TP N°2 en suivant le tutoriel fourni en annexe 3. Appeler le projet, par exemple, TP6_Simple_BT. Le compiler

Dans un 1er temps, nous allons créer une application simple permettant d'activer et de désactiver le Bluetooth du *device* (du smartphone) s'il en possède. La totalité du TP se déroule dans le fragment de la « Page1 »

→ Créer une IU avec un bouton « Activer le Bluetooth » en haut de l'écran et un TextView tout en bas de l'écran. A l'ouverture de l'application, le TextView n'affiche rien. Utiliser le code XML du précédent projet pour gagner du temps (dossier res du projet). Le TP utilise beaucoup l'outil « Toast » pour valider les différents points.

→ Déclarer les objets dans le code de la méthode OnCreateView permettant de manipulation du bouton et du TextView :

```
Button lanceBT = (Button)vue.findViewById(R.id……; // associé au bouton
TextView afficheBT = (TextView) vue.findViewById(R.id……..);// associé au TextView
```

→ Compiler et exécuter l'application pour vérifier.

## 2 Gérer les permissions (autorisations)

Afin de pouvoir utiliser le service Bluetooth fourni par Android, nous devons gérer les permissions qui le concernent. Ceci se fait en deux étapes :

1ere étape : autorisation statique :

→ ajouter les permissions adéquates dans le fichier **AndroidManifest.xml**

Tout est expliqué ici :

https://developer.android.com/guide/topics/connectivity/bluetooth/permissions

Ci-dessous, le maximum des permissions possibles, en tenant compte des versions d'ANDROID…

```
<uses-feature android:name="android.hardware.bluetooth" android:required="false"/>

<!-- Request legacy Bluetooth permissions on older devices. -->
<uses-permission android:name="android.permission.BLUETOOTH"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />

<!-- Needed only if your app uses Bluetooth scan results to derive physical
location. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<!-- Needed only if your app makes the device discoverable to Bluetooth  devices. -
->
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />

<!-- Needed only if your app communicates with already-paired Bluetooth   devices.
-->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
```

2ème étape : vérifier, de façon dynamique que le BLUETOOTH est autorisé par la machine :

On écrit ici un code qui sera exécuté une fois au lancement de l'application et qui vérifie que la machine cible autorise le BLUETOOTH.

→ Au début de Fragment1, déclarer les données ci-dessous :

```
private BluetoothAdapter mBluetoothAdapter; // objets pour manipuler le BT
private BluetoothManager mBluetoothManager;
private PackageManager packageManager;

private Integer interfaceON = 1;  // 4 drapeaux pour valider des étapes
private Integer interfaceOK = 0;
private Integer appareilConnecte = 0;
private Integer autorisationConnexion = 1;
```

→ Le code ci-dessous permet de tester si l'appareil est doté d'une interface BT.Ecrire le code ci-dessous dans la méthode OnCreateView.

```
BluetoothManager mBluetoothManager = (BluetoothManager)
getActivity().getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
if (mBluetoothAdapter == null) {
    // Device doesn't support Bluetooth
    Toast.makeText(getActivity(), "La machine ne possède pas le Bluetooth",
Toast.LENGTH_SHORT).show();
    interfaceOK = 0;
}
else
{
    interfaceOK=1;
    Toast.makeText(getActivity(), "interface BT existe",
Toast.LENGTH_SHORT).show();
}
```

→ Compiler et exécuter l'application pour vérifier. Avec le simulateur, pas de BT, avec une machine ANDROIS cela doit fonctionner.

→ Expliquer le rôle du drapeau « interfaceOK»

# 3 Activer le Bluetooth

## 3.1 Mise en place

Nous ajoutons ici la fonctionnalité ci-dessous :

Au clique sur le bouton, le programme vérifie que le BT est allumé. Si oui, c'est OK. Si non, une autorisation est demandée à l'utilisateur. Si la réponse est positive, c'est OK.

→ Créer un « Listener » pour détecter le « click » sur le bouton sur le bouton « Activer le Bluetooth » et ajouter le code suivant dans la méthode.

```
if (interfaceOK == 1) {
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        Log.i("BTT", "1");

         getActivity().startActivityForResult(enableBtIntent, 10);

        Log.i("BTT", "2");
    } else {
        interfaceON = 1; // le bluetooth est allumé
        Toast.makeText(getActivity(), "Bluetooth allumé",
Toast.LENGTH_SHORT).show();
    }
}
```

Le nombre 10 a été choisi au hasard. Il sera utilisé pour reconnaitre notre demande.
En effet, la méthode getActivity va lancer l'exécution automatique de la fonction « callback » ci-dessous, qu'il faut écrire dans le Fragment 1 :
Le nombre 10 sera le « RequestCode ».
Si le « ResultCode » est différent de 0, le BT est allumé.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode,data);
    Log.i("BT", "onActivityResult, requestCode: " + requestCode + ", resultCode: "
+ resultCode);
    if (resultCode !=0) {
```

```
        interfaceON =1;
    // Ici on pourrait ajouter de code
     }
     else interfaceON=0;
}
```

→ Compiler et exécuter l'application. Vérifier dans la trace d'exécution (Logcat) que le *log* s'affiche bien.
→ La méthode est sans doute annoncée « deprecated ». La rédactrice de ce document n'a pas trouvé plus simple…

# 4 Afficher la liste des appareils connus

Dans cette partie, nous allons améliorer notre application pour afficher la liste des appareils qui sont déjà associés (ou appairés) une fois le BT allumé. Ceci ne signifie pas qu'ils sont connectés.

## 4.1 Mise en place de l'interface

→ Ajouter sur le bord gauche de l'interface

- un bouton « Appareils associés » vers le bas
- une ListView1 pour afficher la liste des appareils déjà associés, au-dessus de ce bouton. L'affichage devra montrer (non de l'appareil+ adresse MAC).

- un bouton tout en bas pour effacer la liste de l'écran

On doit obtenir à peu près ceci :


## 4.2 Atelier : récupération des appareils déjà associés

Pour récupérer la liste des appareils déjà connus de votre machine, Android propose une méthode de sa classe BluetoothAdapter :

```
mBluetoothAdapter.getBondedDevices();
```

La documentation ANDROID, toujours la même, suivie pas à pas, explique bien la procédure : https://developer.android.com/guide/topics/connectivity/bluetooth/find-bluetooth-devices

→ En utilisant la documentation Android (paragraphe « Query paired devices » uniquement), implémenter la recherche des appareils Bluetooth *(devices)* déjà appairés. On utilisera le drapeau `interfaceON` pour ne lancer le processus que si l'on est sûr que le BT est disponible et allumé. Ceci se fera via un « listener » sur le bouton « APPAREILS ASSOCIES ». La liste doit s'afficher dans la « ListView ».

Ces appareils déjà appairés au smartphone sont prêts à communiquer avec lui. Ils peuvent être « vus » par l'application même s'ils ne sont pas présents. Il faut noter qu'il n'y a pas de communication BT mise en place ici, mais uniquement de la lecture d'information interne à la machine.

→ Coder aussi l'appui sur le bouton « EFFACER LISTE ».

On doit obtenir ceci :


# 5 Communication avec le lecteur de code-barres

## 5-1 Configuration du lecteur

Pour illustrer notre TP, nous allons utiliser un lecteur de code-barres de la marque Wasp. Le modèle est le WWS110i ou le WWS10SBR.
Pour que la communication soit possible avec cet appareil, il faut qu'il soit chargé (il fonctionne sur batterie).

→ Il doit être configuré, ceci se fait en scannant des code-barres dans la documentation, selon la démarche ci-dessous. Le lecteur émet 2 « bips » à chaque lecture correcte.

1- Scanner le code-barre de la configuration par défaut (p6)
2- Scanner le code-barre de la communication HID (p7)
3- Scanner le code-barre autorisant la connexion à un nouveau client (Set Connexion) par défaut (p7)
4- Scanner le code pour que le dernier caractère émis soit « CR » (code ASCII 13)
5- Associer l'appareil dans les paramètres BT du téléphone.

Remarque : pour lancer l'association du lecteur, il faut appuyer sur le bouton supérieur, la LED clignote. Quand l'appairage se fait, le lecteur émet 2 « bips » rapprochés.

On peut ensuite utiliser l'appareil pour scanner des codes-barres. Ceux-ci sont transmis sous forme ASCII, les deux derniers caractères étant « CR » de code ASCII 13 et « LF » de code ASCII 10

Exemple :


Figure 2 - Exemple de code-barre

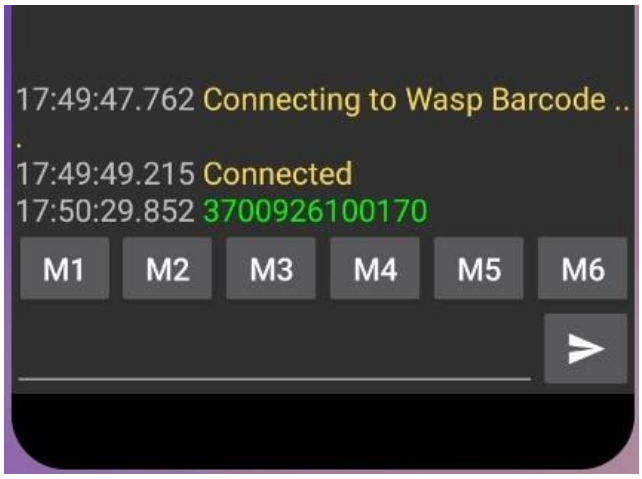
Code transmis ici : 51 48 51 51 52 57 48 48 48 52 53 56 51 13 10

→ Vérifier que le lecteur de code-barre apparait dans la liste des machines appairées et noter son adresse MAC.

→ Si on éteint le Bluetooth, le lecteur se reconnecte en quelques secondes.

*→ Pour tester la bonne réception des codes-barres, installer l'application « Serial Bluetooth » sur un smartphone. Celle-ci est un « terminal ». La donnée envoyée par le lecteur s'affiche dans le terminal. Elle fournit aussi l'adresse MAC de l'appareil. Il faut connecter l'appareil.*

On obtient ceci en scannant un code barre : les caractères en vert représentent le code-barre scanné.



# 6 Lecture des données émises par un appareil Bluetooth – utilisation de « UIThread »

La lecture des données émises par un appareil Bluetooth peut conduire à un « plantage » de l'application si, par exemple, l'appareil avec lequel on souhaite échanger ne répond pas ou si l'échange dure plusieurs secondes comme cela est souvent le cas.
Nous allons utiliser le concept de « UIThread ». Ces tâches, contrairement aux services, sont déclarées dans une activité (ici le fragment1) et sont exécutées, par exemple dans un « listener » suite à l'appui sur un bouton ou un autre évènement. Un mécanisme particulier doit être mis en œuvre pour récupérer dans l'IU les données manipulées dans la tâche.

Voici la syntaxe de cette UIThread, dans sa version 2023. Des recherches sur Internet peuvent conduire à une autre syntaxe. La version d'ANDROID STUDIO utilisée proposera la version adaptée.
Dans le fragment1 (en dehors de « OnCReate »), on déclare et on code la tâche :

```
void nomTache() {

}
```

Dans un « listener », on lance la tâche, là où on veut qu'elle soit exécutée. On peut écrire du code avant de lancer la tâche

```
monBouton.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
    new Thread(this::nomTache).start();
    });
```

Remarque : on ne peut pas utiliser le « Toast » dans une tâche

## 6-1 Préparation de l'interface

→ Créer deux boutons au milieu de l'interface à droite, marqués « CONNECT » et « LIRE CODE-BARRE »


Figure 3 - Mise à jour de l'interface

## 6-2 Connexion du lecteur de Code-barres

→ Dans le listener du bouton « APPAREILS ASSOCIES », (voir ci-dessous et paragraphe 4), récupérer le lecteur de code-barres dans la liste des machines appairées grâce à son adresse MAC. Pour cela, on déclare un objet « deviceTrouve » de type `BluetoothDevice` dans le Frgement1 .

```
private BluetoothDevice deviceTrouve;
```

Puis on modifie le code de la question 4.2

```
if (pairedDevices.size() > 0) {
    for (BluetoothDevice device : pairedDevices) {
     adapter.add(device.getName() + "\n" + device.getAddress());

if (device.getAddress().equals("00:1C:97:1A:4C:F6")) {
// lecteur de code barres
    deviceTrouve = device;
    Log.i("BT", device.getAddress());
        }
    }
}
```

→ Compiler et exécuter pour vérifier que cela fonctionne. Vérifier le logcat.

Nous allons maintenant connecter notre appareil à l'application. La méthode suivie ici est celle proposée dans la documentation :
https://developer.android.com/guide/topics/connectivity/bluetooth/connect-bluetooth-devices

un socket Bluetooth est une connexion établie entre 2 machines (1 client et un serveur)

→ Ecrire le début du code de la tâche (thread) `ConnexionAppareil` qui permet d'établir une connexion entre le téléphone et le lecteur de code-barres. BIEN LIRE les commentaires et poser des questions au professeur !

→ Lancer la tâche dans un listener attaché au bouton « connect ».

```
connect.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
    new Thread(this:: ConnexionAppareil).start();
    });
```

→ Compiler et exécuter pour vérifier que cela fonctionne. On pourra place un filtre sur le « TAG » « BT » pour suivre les LOG dans la fenêtre LogCat. Attention, telle que notre application est écrite, il faut lancer la recherche d'appareils déjà jumelés (pour actualiser la variable `deviceTrouve1` pour que notre programma fonctionne. Si l'application n'a pas été lancée depuis longtemps, il vaut mieux faire une recherche pour vérifier que l'appareil apparait dans l'environnement du téléphone.

→ Ajouter maintenant cette portion de code à la suite dans la tâche « Connexion Appareil » .

```
          BluetoothSocket tmp = null;

                if (interfaceON == 0) {

        Log.i("BT", "Le BT n'est pas prêt");
                } else {
                    // déclarer mmDevice dans l'activité
                    mmDevice = deviceTrouve;   // lecteur de code barre
        "00:1C:97:1E:C7:B2"
                    if (mmDevice == null) {
                        Log.i("BT", "le lecteur de code-barres n'est pas reconnu");

                    } else {


        // les UUID sont des identifiants universels codés sur 128 bits
        // Chaque appareil possède son propre UUID
        // Mais il existe des UUID dédiés
        // Nous utilisons ici celui qui désigne les appareils audio en BT qui
        utilisent le // protocole RFCOM (A2DP_ADVAUDIODIST_UUID)
        // un socket Bluetooth est une connexion établie entre 2 machines
// le code ci-dessous est suggéré par Android Studio pour demander une //
autorisatioon dynamique

                    if (ActivityCompat.checkSelfPermission(getContext(),
BLUETOOTH_CONNECT) != PackageManager.PERMISSION_GRANTED)
                    {
                        Log.i("BT", "pas de permission");
                        return;
                    }// on s'arrête là
                    else
                    {
                        try {
                            tmp =
mmDevice.createRfcommSocketToServiceRecord(UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB"));
                            Log.i("BT", "socket cree");
```

```
                    autorisationConnexion = 1;
                } catch (IOException e) {
                    Log.i("BT", "erreur creation socket");

                    return; // si pb on s'arrete là
                }

// la suite du code ici
}}
```

→ Compiler et exécuter pour vérifier que cela fonctionne. On doit avoir le message « socket cree » dans le LogCat.
→ Déclarer dans le fragment 1 : `private BluetoothSocket mmSocket = null;`

→ Ajouter maintenant à la suite de ce qui précède le code ci-dessous :

```
    if (autorisationConnexion == 1) {
                    mmSocket = tmp;
// le socket est en place

                    // on tente une connexion avec le lecteur de code-barre.
// le constructeur recommande une attente de 12s environ
// pour laisser le temps de l'établissement de la connexion
                    try {

// This is a blocking call and will only return on a
// successful connection or an exception
                        Log.i("BT", "début attente connexion");
                        mmSocket.connect();

                        SystemClock.sleep(12000);  // ce sont des ms
                        Log.i("BT", "fin attente connexion - connexion ok ");
                        appareilConnecte=1;
                    } catch (IOException e) {

                        Log.e("BT", e.getMessage());

                        // ceci permt d'obtenir des infos sur le pb, si pb il y
a
                        try {
                // si la connexion ne se fait pas on ferme le socket
                            mmSocket.close();
                            Log.i("BT", "erreur fermeture socket ");

                        } catch (IOException e2) {
                            Log.i("BT", "impossible de fermer le socket");

                        }
                    }
                }
```

A ce stade, le lecteur de Code-barres est prêt à envoyer des messages au téléphone. Nous allons faire cela dans une autre tâche.
→ Compiler et exécuter pour vérifier que cela fonctionne. Surveiller le Logcat.

## 6-3 Réception d'un code-barre

→ Ecrire les déclarations dans le fragment1, puis le début du code de la tâche (thread) `LireCode()`

qui permet de mettre le lecteur de code-barres en attente d'un message.

```
InputStream tmpIn = null;
InputStream receiveStream = null;
byte[] buffer = new byte[100]; // pour la lecture des données
String resultat="";

Integer tailleMessage;
String nomMessage;
TextView afficheReceptionBT;  // pour afficher le code barre dans l'UI
String finalResultat;  // La chaine de caracteres dans laquelle est mémorisée le
code barre lu
// La chaine de caracteres dans laquelle est mémorisée le code barre lu


// code tache LireCode()
int i;
InputStream tmpIn = null;
// OutputStream tmpOut = null; // non utilisé ici pour le flux dans l'autre sens

// Get the BluetoothSocket input and output streams
Integer lectureOK = 0;
try {
    tmpIn = mmSocket.getInputStream();
    //  tmpOut = mmSocket.getOutputStream();
    receiveStream = tmpIn;
    //  sendStream = tmpOut;  // pour un flux émis non fait ici

    Log.i("BT", "attente code-barre");
    tailleMessage = 0;  // la variable tailleMessage permet de compter les codes
lus. Un code complet envoie 15 caractères
    // les deux derniers sont LF et CR
    do {
        // Read from the InputStream
        numBytes = receiveStream.read(buffer);
        tailleMessage = tailleMessage + numBytes;
        for (i = 0; i < numBytes; i++) {
            if ((buffer[i] != 10) && (buffer[i] != 13)) {
                resultat = resultat + buffer[i];
                Log.i("BT", "partiel " + resultat);
                Log.i("BT", "partiel " + numBytes.toString());
            }
            if ((buffer[i] == 10) || (buffer[i] == 13)) {
                lectureOK = 1;
            }
        }
    }
    while (lectureOK!=1) ;

    if (lectureOK == 1) {
        //lireCode.setText(resultat);
 Log.i("BT", "resultat final " + resultat);
    }

}
// fin 2ème try
catch(Exception e){
    Log.e("BT", "fin de message", e);
    return;
}
```

→ Ajouter le code qui lance la tâche quand on appuie sur le bouton « LIRE CODE_BARRE »

→ Compiler et exécuter pour vérifier que cela fonctionne. Scanner un code-barre. Celui-ci doit apparaitre dans le LogCat. Voilà un exemple (Figure 4) de ce que l'on peut obtenir avec le code-barre de la figure 2. Comprendre ce résultat. La table ASCII est donnée en annexe.

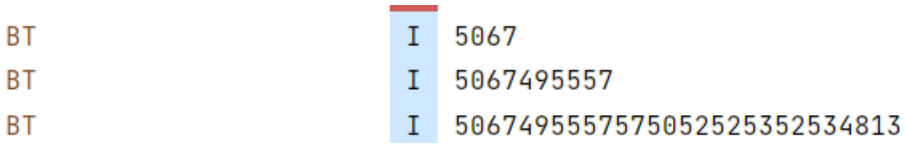



Figure 4 - LogCat de la lecture d'un code-barre

Si on arrêt d'utiliser le lecteur, il se déconnecte.

## 6-4 Récupération du code-barre dans l'IU.

Pour l'instant, le code-barre est visible dans le LogCaT. Il restera à voir comment le récupérer dans le fragment, par exemple en l'affichant dans un TextView.

Nous devons placer la chaine de caractères récupérée dans un « message », et passer celui-ci à l'IU.

On utilisera pour cela un « handler » qui servira de vecteur.

Ceci sera à faire….

Il faudra aussi étudier comment fournir la donnée au 2ème fragment…

# ANNEXE 1 : Manipulation d'une ListView

On crée sur l'interface une ListView ce qui se traduit par du code XML

```
<ListView
    android:id="@+id/listAppConnus"
    Etc…..
```

Dans le code de l'application, on déclare un objet de type LIstView

```
ListView malistView = (ListView) findViewById(R.id.listAppConnus);
```

on déclare un objet de type ArrayAdapter, avec un modèle de liste, ici on a choisi le modèle « simple_list_item_1 »

```
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, new ArrayList<String>());
```

on associe le tableau à la liste, ce qui permettra de la remplir avec des chaines de caractères avec une taille adaptable

```
listView.setAdapter(adapter);
```

Plus loin dans le programme, si on veut afficher du texte dans la liste, on écrira, par exemple. :

```
adapter.add("Bonjour" + "\n" + toto);
```

à condition que "toto » soit une chaine de caractères. "\n" engendra une nouvelle ligne.

# ANNEXE 2 : Table ASCII

Dec = Decimal; Hex = Hexadecimal; Char = Character

| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | Null | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | Start of heading | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | Start of text | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | End of text | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | End of transmit | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | Enquiry | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | Acknowledge | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | Audible bell | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | Backspace | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | Horizontal tab | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage return | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data link escape | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg. acknowledge | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End trans. block | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitution | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |

# ANNEXE 2 : Dupliquer un projet avec Android studio

Ce tutoriel montre comment importer un projet avec Android Studio. Il a été écrit en utilisant cette adresse : https://chezdom.net/etu/dupliquer-un-projet-android/

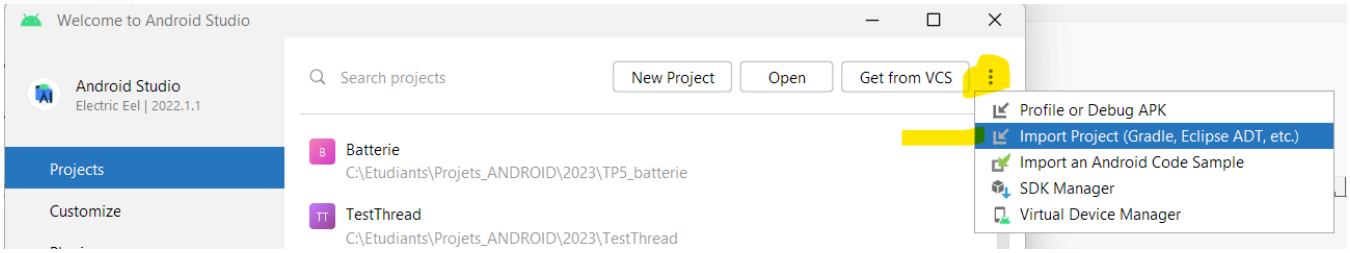
Cela ne marche pas forcément du 1er coup …

1- Faire une copie « recursive » du dossier du projet du TP2 et lui donner un nom qui corresponde au nouveau projet :

Lancer le « poweshell » et taper

```
PS C:\Users\joell> cp -R c:\Etudiants\Projets_Android\nomprojetTP2 c:\Etudiants\Projets_Android\nouveauNom
```

2- Lancer Android Studio et dans l'écran de choix des projets existants, choisir « UImport Project » (Figure ci-dessous)




3- Suivre les indications ci-dessous :

Pour le 3ème point, il faut mettre à jour le nom du package dans l'activité principale et les deux fragments.

- Renommez le package : dans le navigateur de projet (vue en arbre, sur la gauche), dans la branche java, sélectionnez avec la souris le nom du package (net.monurl.chemin.projet1), et utilisez appliquer le menu contextuel `Refactor/Rename`. Normalement un warning doit vous indiquer que ce package correspond à plusieurs répertoires (directories), choisissez `Rename package` (par exemple on pourra mettre « net.monurl.chemin.projet2 » à la place de « …projet1 »), de façon à ce que le changement de nom soit reporté partout à c'est nécessaire (ressources, code java, manifest). Attention il y a encore une confirmation à donner (dans un cadre en bas de l'interface).

- Ouvrir le fichier `AndroidManifest.xml`, et vérifiez le nom du package :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.monurl.chemin.projet1">
```

- Si besoin aller dans le fichier `strings.xml` pour modifier manuellement le nom de l'appli (la chaîne de caractère), ou bien un numéro de version (c'est utile de le faire
afin d'être sûr de déboguer le bon code).

- Enfin il faut mettre à jour 2 scripts gradle (vous les trouvez dans le navigateur de projet :

- dans le script gradle build.gradle (Module: app), changer l'applicationId (sinon la nouvelle app prendra la place de l'ancienne dans votre téléphone)

- le nom de projet dans le settings.graddle (il faut alors faire un « sync » du gradle, studio vous propose de le faire lorsque vous éditez ce fichier). Lorsque vous modifiez cette valeur, Gradle vous propose de faire un Project Sync, faites le en cliquant sur 'Sync Now'.

- Aller dans le menu `Build` et lancer `Clean project` afin de tout remettre à jour, puis « Rebuild Project ».