Formation SIN

Utilisation du logiciel Basic4Android

Formateur : Pierre Aguillon

académie d'Orléans-Tours

PARTIE 4 : PROGRAMME 4

Objectifs :

- **Mettre en œuvre une communication en TCP Client ;**
- **Utiliser le logiciel gratuit Hercules ;**
- **Découvrir la librairie RandomAccessFile;**

Programme 4 :

Ce quatrième programme va nous permettre de rendre communicante notre tablette via le réseau Wifi en utilisant le protocole TCP.

Pour pouvoir mettre en œuvre les programmes, il nous faut l'environnement suivant :

- Une tablette connectée en Wifi sur un réseau et dont on connait l'adresse IP attribuée par le serveur DHCP ;
- Un ordinateur connecté (peu importe le type de connexion) sur le même réseau et dont on connait l'adresse IP attribuée par le serveur DHCP (IPCONFIG) ;
- Le logiciel Hercules de chez HWgroup que l'on peut télécharger ici http://www.hw-group.com/products/hercules/index_en.html ;

En TCP le principe est plus complexe quand UDP. Il faut d'abord définir si votre tablette sera client ou serveur TCP. Généralement, dans nos applications, elle est « Client ». On va donc limiter l'étude à cette fonctionnalité.

Pour cela il faut se remémorer le principe de la connexion en TCP :

- Le serveur créée un port TCP (plus exactement un socket) puis il le met en écoute – ici c'est le logiciel Hercules qui va s'en charger ;
- Le client fait une demande de connexion ;
- Le serveur valide cette demande ;
- L'échange des données est effectif ;
- Le client fait une demande de fermeture du socket ;
- Le serveur ferme le socket et le repositionne en écoute pour une future connexion ;

Dans les exercices, il faudra penser à bien changer les adresses IP en positionnant celles correspondant à votre configuration. Elles seront surlignées en rouge.

Créez un nouveau programme que vous appellerez « programme n4 ».

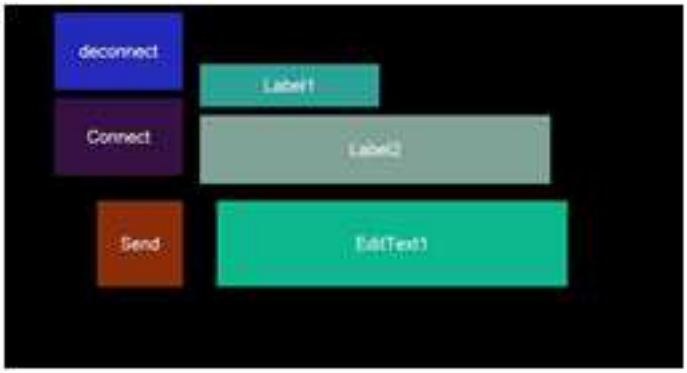
L'utilisation du protocole TCP nécessite l'utilisation des librairies **NetWork** et **RandomAccessFile**. Cochez-les dans votre liste de librairie.

Lancez le Designer.

On va construire une feuille contenant :

- 1 bouton : Name=send ; Text=Send
- 1 bouton : Name=connect ; Text=Connexion
- 1bouton : Name=deconnect ; Text=Déconnexion
- 1 label : Name=label1 ;
- 1 EditText : Name=edittext1 ;

Placez de la manière suivante :



Générez les membres, seuls les **click** sur les trois boutons nous sont utiles. Sauvegardez votre feuille sous le nom **layout1** et fermez le Designer.

Les fonctionnalités de notre programme devront être les suivantes :

- Le texte du **label1** prendra la valeur « connecté » si le client est connecté et « non connecté » si le client ne l'est pas ;
- Le bouton **connect** doit permettre la connexion ;
- Le bouton **deconnect** la déconnexion ;
- Le bouton **send** l'envoi de deux octets ;
- Lorsque le client est connecté seuls les boutons **deconnect** et **send** doivent être visibles ;
- Lorsque le client est déconnecté seul le bouton **connect** doit être visible.

En premier lieu on va placer les lignes de codes exécutées lors du lancement de l'application sous Android. Dans ce cas la connexion n'est pas réalisée, ce qui explique les lignes de codes encadrées en rouge *(note : on aurait pu les placer dans le Designer)*

```
23 Sub Activity_Create(FirstTime As Boolean)
24     Activity.LoadLayout("layout1")
25     Label1.text="Non connecté"
26     Connect.Visible=True
27     deconnect.Visible=False
28     Send.Visible=False
29     EditText1.Visible=False
30 End Sub
```

L'utilisation en client du protocole TCP sous B4A se passe en cinq étapes : la création du socket, son initialisation (adresse IP, port, la taille mémoire allouée au buffer), la connexion du socket (en TPC on travaille en mode connecté), l'échange des données (si le socket est connecté), et enfin la fermeture du socket.

La création du socket : on définit la variable **Mon_socket** comme étant un **Socket** (contrairement à l'UDP où l'on doit préciser **UDPSocket**, le type **Socket** seul correspond au TCP). Lorsque l'on utilise un Socket, on dispose de deux propriétés importantes :

- Socket.inputStream : c'est le flux de données entrantes dans le socket (la réception de données)
- Socket.outputStream : c'est le flux sortant (l'émission de données) ;

```
Sub Globals
    'These global variables will be redeclared each time the activity is created.
    'These variables can only be accessed from this module.
    Dim Mon_Socket As Socket
    Dim flux As AsyncStreams
    Dim Send As Button
    Dim Label1 As Label
    Dim Connect As Button
    Dim deconnect As Button
    Dim EditText1 As EditText
    Dim bc As ByteConverter
    Dim Label2 As Label
    End Sub
```

L'initialisation du socket et la tentative de connexion : C'est l'appui sur le bouton Connect qui doit initialiser le Socket et faire une tentative de connexion sur le serveur dont on connait l'adresse IP (ici celle de votre PC)

```
Sub Connect_Click
    Mon_Socket.Initialize("Mon_Socket")
    Mon_Socket.Connect("192.168.1.90",5500,500)
End Sub
```

S'il y a une tentative de connexion, un évènement se déclenche : il s'agit de la fonction **Mon_Socket_Connected(Succes as Boolean)**. Celle-ci renvoie une variable (**Succes**) qui peut prendre deux valeurs **True** ou **False**, si la connexion est effective ou pas. Suivant ce résultat on va modifier la propriété de nos éléments pour répondre aux exigences du programme.

```
Sub Mon_Socket_Connected(Succes As Boolean)
    If Succes=True Then
    Label1.text="Connecté"
    Connect.Visible=False
    deconnect.Visible=True
    Send.Visible=True
    EditText1.Visible=True
    flux.Initialize(Mon_Socket.InputStream,Mon_Socket.OutputStream,"flux")
    Else
    Label1.text="Non connecté"
    Connect.Visible=True
    deconnect.Visible=False
    Send.Visible=False
    EditText1.Visible=False
    End If
End Sub
```

L'échange de données : c'est ici que la librairie **RandomAccessFile** va nous être utile.

Cette dernière propose un objet, nommé **AsyncStreams**. **AsyncStreams** vous permet de lire des données d'un **InputStream** et d'écrire des données dans un **OutputStream,** sans bloquer votre programme.

Quand de nouvelles données sont disponibles sur l'**InputStream,** l'événement **NewData** est déclenché avec les données.

Quand vous écrivez des données à **l'OutputStream,** elles sont ajoutées à une file d'attente interne et envoyées ensuite en arrière-plan.

AsyncStreams est très utile pour des flux de données lents tels que les réseaux Ethernet ou Bluetooth.

Pour utiliser l'objet **AsyncStreams**, il faut d'abord le définir (1) puis l'associer au flux de données de notre socket lorsque la connexion est effective (2)

```
Sub Globals
    'These global variables will be redeclared each time the activity is created.
    'These variables can only be accessed from this module.
    Dim Mon_Socket As Socket
    Dim flux As AsyncStreams        1
    Dim Send As Button
    Dim Label1 As Label
    Dim Connect As Button
    Dim deconnect As Button
    Dim EditText1 As EditText
    Dim bc As ByteConverter
    Dim Label2 As Label
    End Sub
```

```
Sub Mon_Socket_Connected(Succes As Boolean)
    If Succes=True Then
    Label1.text="Connecté"
    Connect.Visible=False
    deconnect.Visible=True
    Send.Visible=True
    EditText1.Visible=True
    flux.Initialize(Mon_Socket.InputStream,Mon_Socket.OutputStream,"flux")    2
    Else
    Label1.text="Non connecté"
    Connect.Visible=True
    deconnect.Visible=False
    Send.Visible=False
    EditText1.Visible=False
    End If
End Sub
```

Pour l'émission : si le bouton **Send** est visible, c'est qu'une connexion est valide. C'est donc dans l'évènement **Send_Click** que l'on va réaliser l'émission de notre message (ici deux octets).

On va d'abord créer le tableau des données à transmettre (1) ; puis on écrit ces valeurs dans ce flux (2).

```
Sub Send_Click
    Dim emission(2) As Byte
    emission(1)=0x34                1
    emission(0)=0x35

    flux.Write(emission)            2


End Sub
```

*Note : on aurait pu vérifier au début de la fonction si la connexion était bien valide en testant si **Mon_Socket.Connected=True**.*

Pour la réception : si des données nouvelles arrivent sur le **InputStream** associé à notre objet **flux**, un événement est déclenché et la fonction **flux_newdata(buffer()as Byte)** est exécutée.

Les données entrantes sont stockées dans le tableau d'octets défini lors du déclenchement

de la fonction : **buffer().** Pour pouvoir afficher ces données en Ascii dans la propriété **label2.text**, il faut les convertir en chaine de caractères. C'est le rôle de la fonction **BytesToString** où l'on doit préciser : le nom du tableau d'octets (**buffer**) ; à partir de quelle position on commence la conversion (**offset**, ici 0) ; jusqu'à quelle position on fait la conversion (tout le tableau, **buffer.length**) ; le format de conversion (« **ASCII** »).

Le programme donne donc :

```
Sub flux_newdata(buffer()As Byte)
    Dim msg As String
    msg=BytesToString(buffer,0,buffer.length,"ascii")
    Label2.Text=msg
End Sub
```

Reste encore à gérer l'appui sur le bouton poussoir **deconnect** devant assurer la déconnexion du socket et la remise en place des propriétés des objets :

```
Sub deconnect_Click
    Mon_Socket.Close
    Label1.text="Non connecté"
    Connect.Visible=True
    deconnect.Visible=False
    Send.Visible=False
    EditText1.Visible=False
End Sub
```

Compilez votre programme et exécutez.

Sur votre PC, lancez Hercules et sous l'onglet TCP_SERVER saisissez le numéro du port 5500. Votre tablette doit pouvoir échanger des données avec le PC.

On vient de voir comment envoyer des octets. Si l'on veut envoyer du texte que l'utilisateur pourra saisir dans l'objet **EditText**, il va falloir modifier le programme pour réaliser des manipulations de variables.

```
Sub Send_Click
    Dim emission() As Byte
    Dim bc As ByteConverter
    Dim message As String
    message=EditText1.text
    emission=bc.StringToBytes(message,"ascii")

    flux.Write(emission)

End Sub
```

*Note : il faut que la bibliothèque optionnelle **ByteConverter** soit sélectionnée (voir programme 3 en UDP)*