

Ce quatrième programme va vous permettre de rendre communicant votre SmartPhone via le réseau Wifi en utilisant le protocole TCP.

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

- Un SmartPhone connectée en Wifi sur un réseau et dont on connaît l'adresse IP attribuée par le serveur DHCP ;
- Un ordinateur connecté (peu importe le type de connexion) sur le même réseau et dont on connaît 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 qu'en UDP. Il faut d'abord définir si votre SmartPhone 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 crée 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.

1. Créez un nouveau programme que vous appellerez « programme n4 ».
2. L'utilisation du protocole TCP nécessite l'utilisation des librairies **NetWork** et **RandomAccessFile**. Cochez-les dans votre liste de librairie.
3. Lancez le Designer.

Vous allez construire une feuille contenant :

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

4. Placez-les de la manière suivante :



5. 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 votre 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.
6. En premier lieu vous allez 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 TCP on travaille en mode connecté), l'échange des données (si le socket est connecté), et enfin la fermeture du socket.

7. **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) ;

```
9 Sub Globals
10     'These global variables will be redeclared each time the activity is created.
11     'These variables can only be accessed from this module.
12     Dim Mon_Socket As Socket
13     Dim flux As AsyncStreams
14     Dim Send As Button
15     Dim Label1 As Label
16     Dim Connect As Button
17     Dim deconnect As Button
18     Dim EditText1 As EditText
19     Dim bc As ByteConverter
20     Dim Label2 As Label
21 End Sub
```

8. **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 connaît l'adresse IP (ici celle de votre PC)

```
59 Sub Connect_Click
60     Mon_Socket.Initialize("Mon_Socket")
61     Mon_Socket.Connect("192.168.1.90",5500,500)
62 End Sub
```

9. Si 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 vous allez modifier la propriété de vos éléments pour répondre aux exigences du programme.

```

40 Sub Mon_Socket_Connected(Succes As Boolean)
41     If Succes=True Then
42         Label1.text="Connecté"
43         Connect.Visible=False
44         deconnect.Visible=True
45         Send.Visible=True
46         EditText1.Visible=True
47         flux.Initialize(Mon_Socket.InputStream,Mon_Socket.OutputStream,"flux")
48     Else
49         Label1.text="Non connecté"
50         Connect.Visible=True
51         deconnect.Visible=False
52         Send.Visible=False
53         EditText1.Visible=False
54     End If
55 End Sub

```

L'échange de données : c'est ici que la librairie **RandomAccessFile** va vous 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'évènement **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.

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

```

9 Sub Globals
10     'These global variables will be redeclared each time the activity is created.
11     'These variables can only be accessed from this module.
12     Dim Mon_Socket As Socket
13     Dim flux As AsyncStreams
14     Dim Send As Button
15     Dim Label1 As Label
16     Dim Connect As Button
17     Dim deconnect As Button
18     Dim EditText1 As EditText
19     Dim bc As ByteConverter
20     Dim Label2 As Label
21 End Sub

```

```

40 Sub Mon_Socket_Connected(Succes As Boolean)
41     If Succes=True Then
42         Label1.text="Connecté"
43         Connect.Visible=False
44         deconnect.Visible=True
45         Send.Visible=True
46         EditText1.Visible=True
47         flux.Initialize(Mon_Socket.InputStream,Mon_Socket.OutputStream,"flux")
48     Else
49         Label1.text="Non connecté"
50         Connect.Visible=True
51         deconnect.Visible=False
52         Send.Visible=False
53         EditText1.Visible=False
54     End If
55 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 vous allez réaliser l'émission de votre message (ici deux octets).

11. Vous allez d'abord créer le tableau des données à transmettre (1) ; puis vous écrivez ces valeurs dans ce flux (2).

```
83
90 Sub Send_Click
91     Dim emission(2) As Byte
92     emission(1)=0x34
93     emission(0)=0x35
94
95     flux.Write(emission)
96
97
98 End Sub
```

* Note : vous auriez 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 votre 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 chaîne de caractères. C'est le rôle de la fonction **BytesToString** où vous devez 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** »).

12. Le programme donne donc :

```
83
84 Sub flux_newdata(buffer()As Byte)
85     Dim msg As String
86     msg=BytesToString(buffer,0,buffer.length,"ascii")
87     Label2.Text=msg
88 End Sub
89
```

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

```
74 Sub disconnect_Click
75     Mon_Socket.Close
76     Label1.text="Non connecté"
77     Connect.Visible=True
78     disconnect.Visible=False
79     Send.Visible=False
80     EditText1.Visible=False
81 End Sub
```

14. Compilez votre programme et exécutez.
15. Sur votre PC, lancez Hercules et sous l'onglet TCP_SERVER saisissez le numéro du port 5500. Votre SmartPhone doit pouvoir échanger des données avec le PC.
16. Vous venez de voir comment envoyer des octets. Si vous voulez envoyer du texte que l'utilisateur pourra saisir dans l'objet **EditText**, il va falloir modifier le programme pour réaliser des manipulations de variables.


```
57 Sub Send_Click
58     Dim emission() As Byte
59     Dim bc As ByteConverter
60     Dim message As String
61     message=EditText1.text
62     emission=bc.StringToBytes(message,"ascii")
63
64     flux.Write(emission)
65
66 End Sub
67
```

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

Utilisation des librairies simplifiées

Le concept de socket TCP peut être quelque peu déroutant pour nos élèves de STI2D. Aussi pour simplifier la transmission en TCP, j'ai compilé une librairie permettant de rendre transparents certaines étapes.

Vous allez maintenant installer la librairie supplémentaire **TCP_client**.

17. Quittez B4A et placez-vous dans le répertoire des librairies additionnelles de B4A (celui que vous avez créé précédemment).

18. Copiez-y les fichiers **TCP_client.jar** et **TCP_client.xml**.

19. Relancez B4A pour ouvrir votre programme précédent (programme 4) et cochez la librairie additionnelle **TCP_client**.

Afin de ne pas réécrire la globalité du code vous n'allez changer que les parties où l'ajout de la librairie additionnelle « rend les choses plus simples ».

20. Plus besoin de définir de socket et de flux, cela est rendu transparent. Aussi détruisez les deux lignes :

```
9 Sub Globals
10     'These global variables will be redeclared each time the activity is created.
11     'These variables can only be accessed from this module.
12     Dim Mon_Socket As Socket
13     Dim flux As AsyncStreams
14     Dim Send As Button
15     Dim Label1 As Label
16     Dim Connect As Button
17     Dim deconnect As Button
18     Dim EditText1 As EditText
19     Dim bc As ByteConverter
20     Dim Label2 As Label
21 End Sub
```

A supprimer

21. L'appui sur le bouton poussoir Connect doit faire une tentative de connexion sur le serveur dont on connaît l'adresse IP (ici celle de votre PC)

```
61 Sub Connect_Click
62     TCP_client.connexion("192.168.1.96",5500)
63 End Sub
```

22. S'il y a une tentative de connexion avec l'utilisation de la librairie, un événement se déclenche : il s'agit de la fonction **TCP_client.Connected(Succes as Boolean)** remplaçant donc la fonction **Mon_socket.Connected** qui peut être détruite.

```
46 Sub TCP_client_Connected(Succes As Boolean)
47     If Succes=True Then
48         Label1.text="Connecté"
49         Connect.Visible=False
50         deconnect.Visible=True
51         Send.Visible=True
52         EditText1.Visible=True
53     End If
54 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 vous allez réaliser l'émission de votre message.

23. Modifiez les lignes associées à l'évènement Send_Click

```
57 Sub Send_Click
58     Dim emission() As Byte
59     Dim bc As ByteConverter
60     Dim message As String
61     message=EditText1.text
62     emission=bc.StringToBytes(message,"ascii")
63
64     TCP_client.emission(emission)
65
66 End Sub
67
```

24. Pour la réception : si des données nouvelles arrivent un événement est déclenché et la fonction **TCP_client_newdata(buffer()as Byte)** est exécutée. Elle remplace donc la fonction **flux_newdata(buffer()as Byte)**.

```
76 Sub TCP_client_newdata(buffer()As Byte)
77     Dim msg As String
78     msg=BytesToString(buffer,0,buffer.length,"ascii")
79     Label2.Text=msg
80 End Sub
```

25. 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. Une seule ligne est à modifier :

```
67 Sub deconnect_Click
68     TCP_client.fin_connexion
69     Label1.text="Non connecté"
70     Connect.Visible=True
71     deconnect.Visible=False
72     Send.Visible=False
73     EditText1.Visible=False
74 End Sub
```