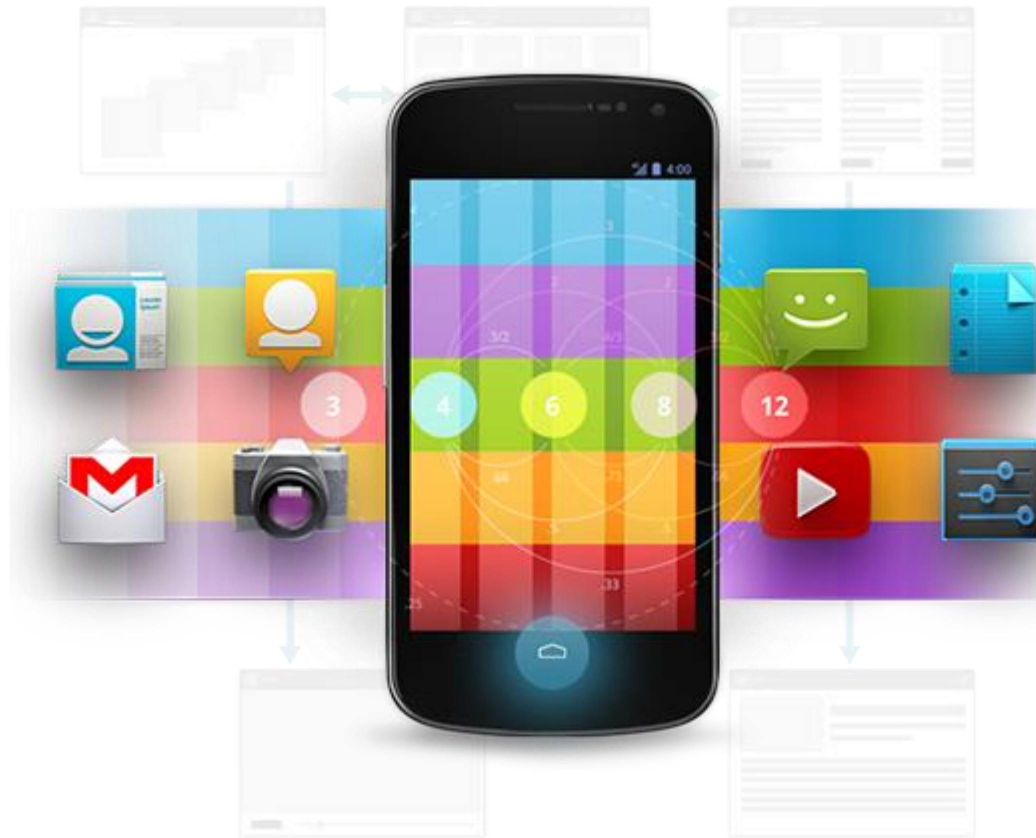


Développement Android (4.3)

Réseau

Benjamin Billet

WARNING

Le contenu de cette présentation est basé sur la documentation anglophone officielle d'Android, diffusée sous licence *Creative Commons Attribution 2.5* :

developer.android.com

La plupart des schémas qui composent ce cours proviennent de cette documentation et sont, par conséquent, soumis à cette même licence.

http://creativecommons.org/licenses/by/2.5/

Au Sommaire !

- Généralités
- Socket/HTTP
- GSM
- Bluetooth
- Wifi P2P & NSD
- NFC

GÉNÉRALITÉS

Réseau

- Ne pas oublier les droits spécifiques.

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

- Il est possible de vérifier l'état du réseau ainsi que le type des connexions actives.

```java
ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);

NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConnected = networkInfo.isConnected();

networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConnected = networkInfo.isConnected();

networkInfo = connMgr.getActiveNetworkInfo();
boolean isOnline = (networkInfo != null && networkInfo.isConnected());
```

Réseau

- Lorsque le système est en mode strict (par défaut), manipuler le réseau dans le main thread provoque une exception !
  - Utiliser des threads ou des AsyncTask (mais attention à bien gérer les runtime changes).
  - Utiliser des services.
- Tout comme le CPU, les interfaces réseaux sont mises en pause lorsque le téléphone passe en mode veille. Il est toutefois possible d'acquérir un lock spécifique au WiFi (haute performance, complet et scan only).

```
WifiManager wm = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
wifiLock = wm.createWifiLock(WifiManager.WIFI_MODE_FULL , "MyWifiLock");
wifiLock.acquire();
...
wifiLock.release();
```

Réseau

- Les changements dans le réseau sont broadcastés par le système :
  - Action : ConnectivityManager.CONNECTIVITY_ACTION (android.net.conn.CONNECTIVITY_CHANGE)
  - Extras : ConnectivityManager.EXTRA_NO_CONNECTIVITY = true, si déconnexion.

```java
@Override
public void onReceive(Context context, Intent intent)
{
    ConnectivityManager conn = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = conn.getActiveNetworkInfo();
    if (networkInfo != null)
    {
        if (networkInfo.getType() == ConnectivityManager.TYPE_WIFI)
            ...
        else if (networkInfo.getType() == ConnectivityManager.TYPE_MOBILE)
            ...
        else
            ...
    }
    else
        ...
}
```

RÉSEAU

- En WiFi, le système broadcaste aussi un évènement après la découverte des réseaux :
  - Action : WifiManager. SCAN_RESULTS_AVAILABLE_ACTION (android.net.wifi.SCAN_RESULTS)

```
@Override
public void onReceive(Context context, Intent intent)
{
    WifiManager connManager = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
    List<ScanResult> scanResults = connManager.getScanResults();
}
```

- Ne pas oublier de déclarer les droits associés, mais aussi d'indiquer que l'application utilise des fonctionnalités WiFi.

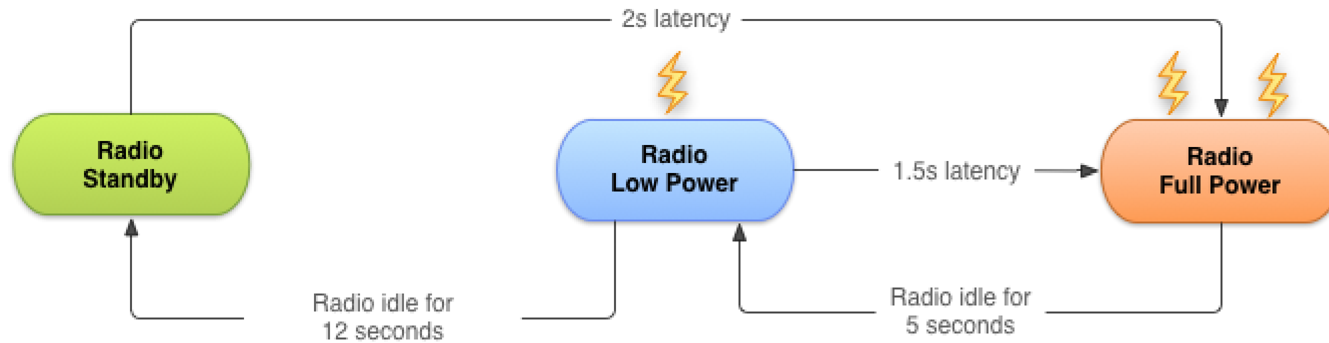
```
<manifest ...>
    <uses-feature android:name="android.hardware.wifi" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    ...
</manifest>
```

Réseau

- Pour économiser l'énergie consommée par le réseau :
  - Préférez les transferts via Wifi (détecter le type de connexion avant d'effectuer certaines actions).
  - Si polling régulier, laissez l'utilisateur choisir la fréquence en fonction de ses besoins et utilisez les alarmes inexactes.
  - Utilisez le cache pour sauver les données, au lieu de les retélécharger.
  - En cas d'absence de connexion (ou d'échec), utilisez un délai croissant entre chaque tentatives :

```
private void retryIn(long interval)
{
    boolean success = checkConnectivity();
    if (success == false)
    {
        interval = interval * 2;
        if (interval > MAX_RETRY_INTERVAL)
            interval = MAX_RETRY_INTERVAL;
        retryIn(interval);
    }
}
```

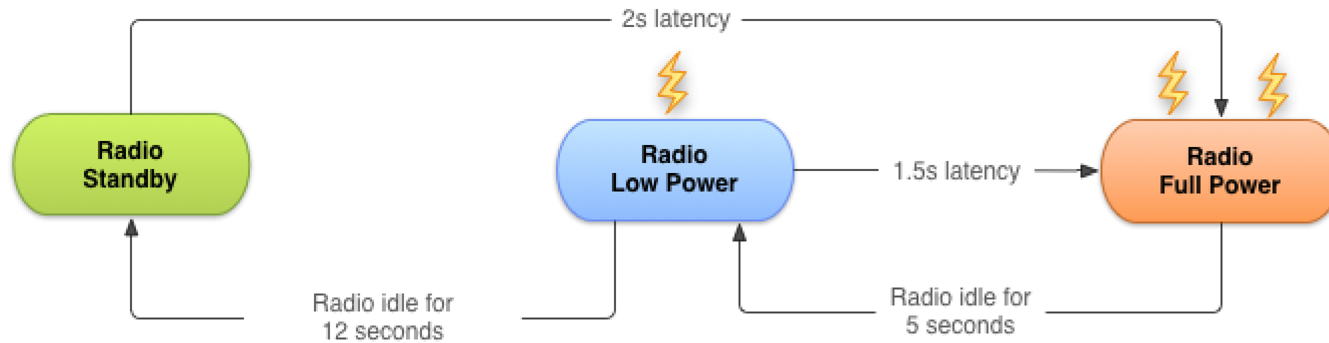
RÉSEAU



- Une pile réseau typique possède trois états :
  - Full power : Connexion active, le débit est maximal.
  - Low power : La pile réseau est en veille.
  - Standby : Aucune connexion active.
- Le temps d'inactivité entre chaque chaque état varie selon le réseau mobile (2G, 3G, WiFi, etc.).
- Chaque connexion force le passage de l'état standby à l'état full, ce qui peut coûter non seulement du temps mais aussi beaucoup d'énergie :

  2s de réveil + 5s en full + 12s low = 20s / connexion.

RÉSEAU



Une bonne pratique consiste donc à grouper les connexions (téléchargements ou envois) lorsque cela est possible.

De même, télécharger plus de données en une seule fois est préférable à télécharger moins de données en plusieurs fois.

l'état full, ce qui peut coûter non seulement du temps mais aussi beaucoup d'énergie :

2s de réveil + 5s en full + 12s low = 20s / connexion.

SOCKET/HTTP

Un retour sur les socket

- Le classique package java.net.

```
BlockingQueue<Runnable> queue = new LinkedBlockingQueue<Runnable>();
ThreadPoolExecutor pool = new ThreadPoolExecutor(20, 100, 5L, TimeUnit.SECONDS, queue);

ServerSocket server = new ServerSocket(8080);
while(shouldStop() == false)
{
    Socket client = server.accept(); // blocking
    pool.execute(new ProcessingTask(client));
}
server.close();

private static class ProcessingTask implements Runnable
{
    private Socket sock;
    public ProcessingTask(final Socket sock)
    {
        this.sock = sock;
    }

    @Override
    public void run()
    {
        BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
        PrintStream out = new PrintStream(sock.getOutputStream());
        message = in.readLine(); // blocking
        out.println(message);
        sock.close();
    }
}
```

Un retour sur les socket

- Le classique package java.net.

```
Socket socket = new Socket("localhost", 8080);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

out.println("Write something"); // send request

// get response
String tmp;
while((tmp = in.readLine()) != null)
{
    ...
}

socket.close();
```

Un retour sur les socket

- Socket non bloquantes avec java.nio.

```
Selector selector = Selector.open();
ServerSocketChannel serverChannel = ServerSocketChannel.open();
serverChannel.configureBlocking(false);
serverChannel.socket().bind(new InetSocketAddress(8080));
serverChannel.register(selector, SelectionKey.OP_ACCEPT);

while (shouldStop() == false)
{
    selector.select(); // blocking (selectNow is non-blocking)
    Iterator<SelectionKey> it = selector.selectedKeys().iterator();
    while (it.hasNext())
    {
        SelectionKey key = (SelectionKey) it.next();
        it.remove();

        processKey(key);
    }
}

selector.close();
serverChannel.close();
```

Un retour sur les socket

```
public void processKey(SelectorKey key)
{
    if (key.isAcceptable())
    {
        ServerSocketChannel server = (ServerSocketChannel) key.channel(); // = serverChannel
        SocketChannel client = server.accept();
        client.configureBlocking(false);
        client.register(selector, SelectionKey.OP_READ);
    }
    else if (key.isReadable())
    {
        SocketChannel client = (SocketChannel) key.channel();
        StringBuilder data = new StringBuilder();
        ByteBuffer buffer = ByteBuffer.allocate(1024);

        int len = client.read(buffer);                     // 0 = nothing to read, should continue
        while (len > 0 || buffer.remaining() == 0)         // < 0 = end of transmission, should close
        {                                                  // > 0 = data read
            buffer.flip();
            data.append(Charset.forName("UTF8").newDecoder().decode(buffer).array());
            buffer.clear();
            len = client.read(buffer);
        }

        if (len < 0)
        {
            key.cancel();
            client.close();
        }
        ...
    }
}
```

Un retour sur les socket

- Socket non bloquantes avec java.nio.

```
SocketChannel client = SocketChannel.open();
client.configureBlocking(false);
client.connect(new InetSocketAddress("localhost", 8080));

Selector selector = Selector.open();
client.register(selector, SelectionKey.OP_CONNECT);

while (selector.select(500) > 0)
{
    Iterator<SelectionKey> it = selector.selectedKeys().iterator();
    while (it.hasNext())
    {
        SelectionKey key = (SelectionKey) it.next();
        it.remove();

        if (key.isConnectable())
        {
            SocketChannel channel = (SocketChannel) key.channel();
            if (channel.isConnectionPending())
                channel.finishConnect();

            ByteBuffer buffer = ByteBuffer.wrap("Some Text".getBytes());
            channel.write(buffer);
            ...
        }
    }
}
```

HTTP

- La classe Java HttpURLConnection.

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap>
{
    @Override
    protected Bitmap doInBackground(String... urls)
    {
        URL url = new URL(urls[0]);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000);
        conn.setConnectTimeout(15000);
        conn.setRequestMethod("GET");
        conn.setDoInput(true);
        conn.connect();

        int response = conn.getResponseCode();
        InputStream is = conn.getInputStream();
        Bitmap bitmap = BitmapFactory.decodeStream(is);
        is.close();

        return bitmap;
    }

    @Override
    protected void onPostExecute(Bitmap result)
    {
        ImageView imageView = (ImageView) findViewById(R.id.image_view);
        imageView.setImageBitmap(result);
    }
}
```

DownloadManager

- Gère automatiquement les téléchargements dans un processus du système.
- Idéal pour les gros téléchargements.
- Affiche les téléchargements sous forme de notifications.
- Permet d'accéder aux téléchargements en cours à la manière d'un Content Provider.

DownloadManager : new download

```
BroadcastReceiver onComplete = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context ctxt, Intent intent)
    {
        long receivedID = intent.getLongExtra(DownloadManager.EXTRA_DOWNLOAD_ID, -1L);
    }
};
BroadcastReceiver onNotificationClick = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context ctxt, Intent intent)
    {
        long[] receivedIDs = intent.getLongArrayExtra(DownloadManager.EXTRA_NOTIFICATION_CLICK_DOWNLOAD_IDS);
    }
};

DownloadManager mgr = (DownloadManager) getSystemService(DOWNLOAD_SERVICE);
registerReceiver(onComplete, new IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE));
registerReceiver(onNotificationClick, new IntentFilter(DownloadManager.ACTION_NOTIFICATION_CLICKED));

mgr.enqueue(new DownloadManager.Request(uri)
         .setAllowedNetworkTypes(DownloadManager.Request.NETWORK_WIFI | DownloadManager.Request.NETWORK_MOBILE)
         .setAllowedOverRoaming(false)
         .setTitle("My File")
         .setDescription("Something useful")
         .setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS, "myfile.mp4"));
```

DownloadManager : Query downloads

```
DownloadManager mgr = (DownloadManager) context.getSystemService(Context.DOWNLOAD_SERVICE);

DownloadManager.Query query = new DownloadManager.Query();
// setFilterById is available too
query.setFilterByStatus(
    DownloadManager.STATUS_PAUSED  |
    DownloadManager.STATUS_PENDING |
    DownloadManager.STATUS_RUNNING |
    DownloadManager.STATUS_SUCCESSFUL);

Cursor cur = mgr.query(query);
int col = cur.getColumnIndex(DownloadManager.COLUMN_LOCAL_FILENAME);
for(cur.moveToFirst(); cur.isAfterLast() == false; cur.moveToNext())
{
    String localFileName = cur.getString(col));
    ...
}
cur.close();
```

GSM

SMS

- Pour envoyer des SMS, deux méthodes :
  - Utilisation de l'application SMS via un Intent.

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "le corps de mon sms");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

  - Utilisation du SmsManager.

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

```
SmsManager smsMgr = SmsManager.getDefault();
smsMgr.sendTextMessage(
    "0102030405",// destination address
    null,        // service center address (null = default)
    "mon sms !", // sms content
    null,        // an intent that will be broadcasted when the sms is sent (or failed)
    null         // an intent that will be broadcasted when the sms is delivered
);

List<String> parts = smsMgr.divideMessage("... a very big message !");
smsMgr.sendMultipartTextMessage("0102030405", null, parts, null, null);
```

TelephonyManager

- Permet d'obtenir des informations sur :
  - L'état de la connexion data (déconnectée, dormante, réception, envoi, etc.).
  - Le réseau (type, pays, opérateur, roaming, etc.)
  - La carte SIM (ID, opérateur, état : présence, PIN/PUK requis, bloquée).
  - Le device (type, IMEI, IMSI).
  - Le call state (en appel, en attente, sonnerie, etc.).
  - Les cellules disponibles (ID, puissance du signal, position, etc.).

TelephonyManager

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

TelephonyManager telMgr = (TelephonyManager) Context.getSystemService(Context.TELEPHONY_SERVICE);
List<CellInfo> cells = telMgr.getAllCellInfo();
if (cells != null) // not supported / no cells
{
    for (CellInfo cell : cells)
    {
        // CellInfoCdma, CellInfoGsm, CellInfoLte or CellInfoWcdma
        if (cell instanceof CellInfoGsm)
        {
            CellInfoGsm gsmCell = (CellInfoGsm) cell;
            CellSignalStrength strength = gsmCell.getCellSignalStrength();
            int db = strength.getDbm();
            int asu = ((CellSignalStrengthGsm)strength).getAsuLevel();

            CellIdentityGsm id = gsmCell.getCellIdentity()
            int cid = id.getCid(); // gsm cell identity
            int lac = id.getLac(); // location area code
            ...
        }
        else if (cell instanceof CellInfoCdma)
        {
            CellInfoCdma cdmaCell = (CellInfoCdma) cell;
            CellIdentityCdma id = cdmaCell.getCellIdentity();
            int latitude = id.getLatitude();
            int longitude = id.getLongitude();
        }
        ...
    }
}
```

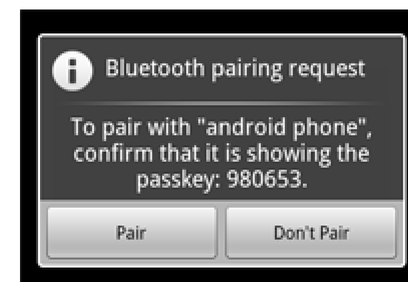
TelephonyManager

```
TelephonyManager telMgr = (TelephonyManager) Context.getSystemService(Context.TELEPHONY_SERVICE);
telMgr.listen(new PhoneStateListener()
{
    @Override
    public void onCallStateChanged(int state, String incomingNumber)
    {
    }
    @Override
    public void onCellInfoChanged(List<CellInfo> cellInfo)
    {
    }
    @Override
    public void onCellLocationChanged(CellLocation location)
    {
    }
    @Override
    public void onDataActivity(int activity)
    {
    }
    @Override
    public void onDataConnectionStateChanged(int state)
    {
    }
    @Override
    public void onSignalStrengthChanged(SignalStrength signalStrength)
    {
    }
});
```

BLUETOOTH

Bluetooth

- Deux périphériques bluetooth doivent s’associer (pairing) avant de communiquer.
  - Découverte (un device peut découvrir tous les autres devices bluetooth à proximité, si ceux-ci sont configurés pour répondre).
  - Utilisation de l’adresse MAC.
- Le pairing nécessite un échange de clé manuel.



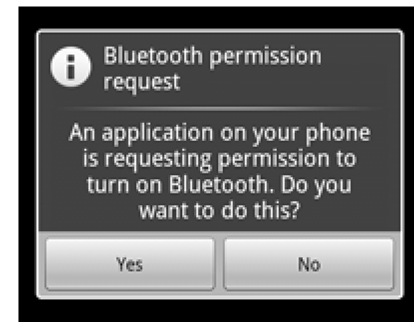
- Un canal peut alors être ouvert entre les deux appareils (attention : une seule connexion à la fois par canal).

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

BluetoothAdapter

- Le BluetoothAdapter gère l’ensemble des opérations Bluetooth.
- Si le Bluetooth n’est pas activé, l’application peut demander à l’utilisateur de le faire directement, avec un Intent.
  - onActivityResult() avec RESULT_OK signifie que l’utilisateur a accepté.
  - RESULT_CANCELED sinon.

```
BluetoothManager btMgr = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
BluetoothAdapter btAdapter = btMgr.getAdapter();
if (btAdapter == null)
    // the device does not support Bluetooth

if (btAdapter.isEnabled() == false)
{
    Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(intent, 42);
}
```



BluetoothAdapter

- L'application peut se tenir informé des changements d'état du Bluetooth, au moyen d'un receiver :
  - Action : BluetoothAdapter.ACTION_STATE_CHANGED (android.bluetooth.adapter.action.STATE_CHANGED).
  - Extras : BluetoothAdapter.EXTRA_STATE et BluetoothAdapter.EXTRA_PREVIOUS_STATE :
    - BluetoothAdapter.STATE_TURNING_ON.
    - BluetoothAdapter.STATE_ON.
    - BluetoothAdapter.STATE_TURNING_OFF.
    - BluetoothAdapter.STATE_OFF.

DÉCOUVERTE

- La découverte est asynchrone et consiste.
- Seuls les devices explicitement définis comme “discoverable” répondent à la requête de découverte.
- L’application peut demander à l’utilisateur de rendre le device discoverable pour un temps limité (120s par défaut).
  - onActivityResult() avec RESULT_OK signifie que l’utilisateur a accepté.
  - RESULT_CANCELED sinon.
  - Rendre un device discoverable active automatiquement le Bluetooth si celui-ci est éteint.

```
if (btAdapter.getScanMode() != BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)
{
    Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    intent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
    startActivity(intent);
}
```

DÉCOUVERTE

- L'application peut se tenir informé des changements de discoverability :
  - Action : BluetoothAdapter.ACTION_SCAN_MODE_CHANGED (android.bluetooth.adapter.action.SCAN_MODE_CHANGED).
  - Extras : BluetoothAdapter.EXTRA_SCAN_MODE et BluetoothAdapter.EXTRA_PREVIOUS_SCAN_MODE :
    - BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE : discoverable et ouvert aux connexions.
    - BluetoothAdapter.SCAN_MODE_CONNECTABLE : non discoverable mais ouvert aux connexions
    - BluetoothAdapter.SCAN_MODE_NONE : non discoverable et fermé aux connexions.

Découverte

- Le processus de découverte prend une vingtaine de seconde (découverte et récupération des noms Bluetooth).
- Il s'agit d'un processus assez lourd, qu'il faut éviter de démarrer pendant que des connexions sont actives (perte de bande passante et déconnexion intempestives) :
  - cancelDiscovery() pour annuler la découverte.

```
BroadcastReceiver receiver = new BroadcastReceiver()
{
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            ...
        }
    }
};

// other useful actions : ACTION_DISCOVERY_STARTED, ACTION_DISCOVERY_FINISHED
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(receiver, filter); // don't forget to unregister during onDestroy

btAdapter.startDiscovery();
```

PAIRING

- Le pairing nécessite d'utiliser le BluetoothDevice obtenu lors de la découverte, où construit à partir d'une adresse MAC statique :

```
BluetoothDevice device = btAdapter.getRemoteDevice("00:43:A8:23:10:F0");
```

- Le pairing est automatiquement demandé lors de l'ouverture d'un canal avec un autre device.
- Les devices déjà associés par le passé sont sauvegardés par le BluetoothAdapter.

```
Set<BluetoothDevice> pairedDevices = btAdapter.getBondedDevices();
for (BluetoothDevice device : pairedDevices)
{
    String name = device.getName();
    String mac = device.getAddress());
}
```

Serveur

- La communication s'établit sur un mode client-serveur, et doit être gérée ailleurs que dans l'UI thread :
  - Client : BluetoothSocket.
  - Serveur : BluetoothServerSocket.
- Une seule connexion à la fois sur le canal identifié par un nom de service et un ID (qui devra être donné par le client pour se connecter).
- Attention à stopper la découverte lorsqu'une requête est en cours.

```
BluetoothServerSocket server = btAdapter.listenUsingRfcommWithServiceRecord("service name", "uuid");
server.close();

BluetoothSocket client = server.accept(); // blocking
if (client != null)
{
    btAdapter.cancelDiscovery();
    InputStream is = client.getInputStream();
    OutputStream os = client.getOutputStream();
    ...
}
```

Client

- Attention à stopper la découverte lorsqu'une requête est en cours.
- connect() est un appel bloquant dont le timeout est d'environ 12 secondes. La communication doit donc être placée dans un thread, un service ou une AsyncTask.

```
BluetoothDevice aDevice = ...;
BluetoothSocket socket = aDevice.createRfcommSocketToServiceRecord("uuid");
btAdapter.cancelDiscovery();
socket.connect(); // blocking
...
socket.close();
```

PROFIL

- Android gère des profils Bluetooth, c'est-à-dire des configurations particulières destinées à certains usages, services ou périphériques (e.g., Hands-Free, Headset, etc.).
- Pour se connecter, les appareils doivent supporter les mêmes profils.
- Quelques profils sont fournis par le système :
  - Headset (inclus Hands-Free).
  - A2DP (Advanced Audio Distribution Profile).
  - HDP (Health Device Profile).

PROFIL

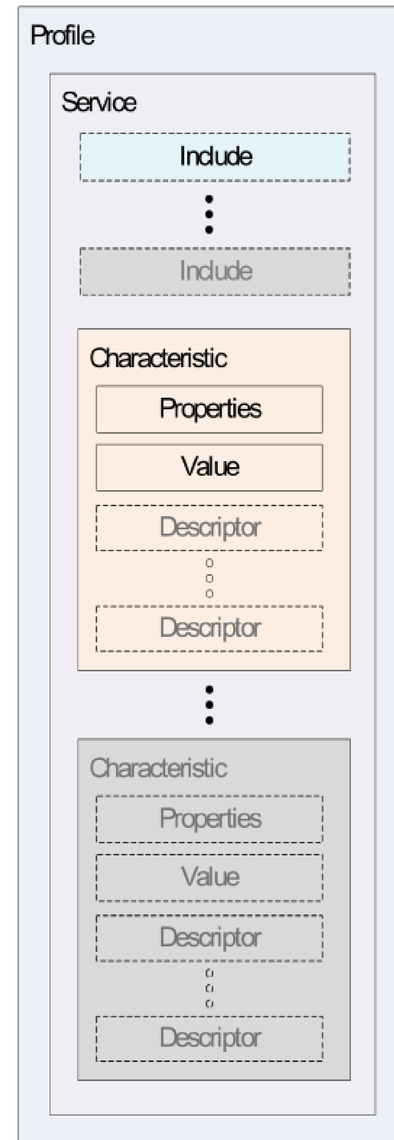
```
public class MyActivity extends Activity implements BluetoothProfile.ServiceListener
{
    private BluetoothHeadset headsetProfile;
    private BluetoothAdapter btAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        ...
        BluetoothManager btMgr = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
        btAdapter = btMgr.getAdapter();
        btAdapter.getProfileProxy(this, this, BluetoothProfile.HEADSET);
    }

    @Override
    public void onDestroy()
    {
        if (headsetProfile != null)
            btAdapter.closeProfileProxy(BluetoothProfile.HEADSET, headsetProfile);
    }

    @Override
    public void onServiceConnected(int profile, BluetoothProfile proxy)
    {
        if (profile == BluetoothProfile.HEADSET)
            headsetProfile = (BluetoothHeadset) proxy;
    }
    @Override
    public void onServiceDisconnected(int profile)
    {
        if (profile == BluetoothProfile.HEADSET)
            headsetProfile = null;
    }
}
```

Bluetooth Low Energy

- Bluetooth LE est une partie du standard Bluetooth 4.0.
- Au prix d'une réduction de portée et de débit, la consommation énergétique est significativement réduite (idéal pour les capteurs Bluetooth).
- Le profil GATT (Generic Attribute Profile) est conçu pour l'échange de petits paquets de données, ou attributs.
- GATT sert de base à une multitude d'autres profils plus spécifiques :
  - BLP (Blood pressure Profile).
  - RSCP (Running Speed and Cadence Profile).
  - PXP (Proximity Profile).
  - CPP (Cycling Power Profile)
  - …

Bluetooth Low Energy

- GATT exploite le protocole ATT (Attribute Protocol), optimisé pour utiliser le minimum de bande passante lors de l'échange d'attribut.
- Trois types d'attributs :
  - Caractéristique : Une valeur, associée à des descripteurs.
  - Service : Une collection de caractéristiques.
  - Descripteur : Une information sur une caractéristique (unité de mesure, plage de valeurs, etc.).
- Chaque attribut est identifié par un UUID.



Découverte

```xml
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

```java
if (getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE) == false)
	// BLE is not supported

BluetoothManager btMgr = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
final BluetoothAdapter btAdapter = btMgr.getAdapter();

final List<BluetoothDevice> found = new ArrayList<BluetoothDevice>();
final BluetoothAdapter.LeScanCallback scanCallback = new BluetoothAdapter.LeScanCallback()
{
	@Override
	public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord)
	{
		found.add(device);
	}
};

Handler handler = ...;
handler.postDelayed(new Runnable()
{
	@Override
	public void run()
	{
		btAdapter.stopLeScan(scanCallback);
	}
}, 10000); // stop scanning after 10s

btAdapter.startLeScan(scanCallback);
```

- Remarque : Il est impossible de faire en même temps une découverte Bluetooth et une découverte BLE.
- startLeScan() peut recevoir un tableau d’UUID pour la recherche de services spécifiques.

Connexion

```
BluetoothGattCallback callback = new BluetoothGattCallback()
{
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState)
    {
        if (newState == BluetoothProfile.STATE_CONNECTED)
            gatt.discoverServices();
        else if (newState == BluetoothProfile.STATE_DISCONNECTED)
            ...
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status)
    {
        if (status == BluetoothGatt.GATT_SUCCESS)
            List<BluetoothGattService> services = gatt.getServices();
    }

    @Override
    public void onCharacteristicRead(BluetoothGatt gatt, BluetoothGattCharacteristic charac, int status)
    {
        if (status == BluetoothGatt.GATT_SUCCESS)
            ...
    }

    ...
};
BluetoothGatt btGatt = device.connectGatt(context, false, callback);
...
btGatt.close(); // close the connection and release the resources
```

Lecture

```
// after service discovery, initial values are read
UUID uuid = service.getUuid();
List<BluetoothGattCharacteristic> characteristics = service.getCharacteristics();
for (BluetoothGattCharacteristic c : characteristics)
{
    List<BluetoothGattDescriptor> descriptors = c.getDescriptors();
    String data = c.getStringValue(0);
    Float f = c.getFloatValue(BluetoothGattCharacteristic.FORMAT_FLOAT, 0);
    Integer i = c.getIntValue(BluetoothGattCharacteristic.FORMAT_UINT16, 0);
    ...
}

// explicit read
BluetoothGattCharacteristic charac = ...;
BluetoothGatt gatt = ...;
BluetoothGattDescriptor desc = ...;
gatt.readCharacteristic(charac);
gatt.readDescriptor(desc);

// in BluetoothGattCallback
@Override
public void onCharacteristicRead(BluetoothGatt gatt, BluetoothGattCharacteristic charac, int status)
{
    if (status == BluetoothGatt.GATT_SUCCESS)
        ...
}

@Override
public void onDescriptorRead(BluetoothGatt gatt, BluetoothGattDescriptor desc, int status)
{
    if (status == BluetoothGatt.GATT_SUCCESS)
        ...
}
```

LECTURE

```java
// receive notifications when a characteristic changes
BluetoothGattCharacteristic charac = ...;
BluetoothGatt gatt = ...;

gatt.setCharacteristicNotification(charac, true);

BluetoothGattDescriptor descriptor = characteristic.getDescriptor(UUID.fromString("..."));
descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
gatt.writeDescriptor(descriptor);


// in BluetoothGattCallback
@Override
public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic)
{
    ...
}
```

WIFI P2P & NSD

WiFi P2P

- Le WiFi P2P (ou WiFi Direct) permet à des appareils de communiquer sans point d'accès.
- Similaire au Bluetooth (découverte, pairing, etc.) mais sur liaison WiFi (débit plus élevé, ainsi que la consommation énergétique).
- Les applications exposent des services qui peuvent alors être découvert dynamiquement.

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

WifiP2PManager

- L'application peut se tenir informé des changements d'état du WiFi Direct, au moyen d'un receiver :
  - Action : WifiP2pManager. WIFI_P2P_STATE_CHANGED_ACTION (android.net.wifi.p2p.STATE_CHANGED).
  - Extras : WifiP2pManager.EXTRA_WIFI_STATE :
    - WifiP2pManager.WIFI_P2P_STATE_ENABLED
    - WifiP2pManager.WIFI_P2P_STATE_DISABLED

```
@Override
public void onReceive(Context context, Intent intent)
{
    int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
    if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED)
        // wifi P2P is enabled
    else
        // wifi P2P is not enabled
}
```

Initialisation

- Lors de l'initialisation, nous obtenons un canal qui servira à la communication avec les autres appareils.
- L'API Wifi Direct d'Android utilise des broadcasts :
  - WIFI_P2P_CONNECTION_CHANGED_ACTION : changement dans la connexion.
  - WIFI_P2P_PEERS_CHANGED_ACTION : découverte de nouveaux appareils.
  - WIFI_P2P_STATE_CHANGED_ACTION
  - WIFI_P2P_THIS_DEVICE_CHANGED_ACTION : changement au niveau d'un appareil (e.g., nouveau nom).

```
private WifiP2pManager p2pMgr;
private Channel channel;

@Override
protected void onCreate(Bundle savedInstanceState)
{
	...
	p2pMgr = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
	channel = p2pMgr.initialize(this, getMainLooper(), null);
	...
}
```

DÉCOUVERTE

- Lorsque la découverte est terminée, WIFI_P2P_PEERS_CHANGED_ACTION est broadcasté.

```
p2pManager.discoverPeers(channel, new WifiP2pManager.ActionListener()
{
     @Override
     public void onSuccess()
     {
     }

     @Override
     public void onFailure(int reasonCode)
     {
     }
});

...

@Override
public void onReceive(Context context, Intent intent)
{
     p2pMgr.requestPeers(channel, new WifiP2pManager.PeerListListener()
     {
          @Override
          public void onPeersAvailable(WifiP2pDeviceList peers)
          {
               Iterator<WifiP2pDevice> it = peers.getDeviceList().iterator();
          }
     });
}
```

Connexion

- Les appareils communiquent au sein de groupes, qui sont rejoint par de nouveaux appareils ou crées automatiquement.
- Le propriétaire du groupe prend alors le rôle de serveur.
- Tout comme pour le bluetooth, rejoindre un groupe déclenche une alerte pour l'utilisateur.

```
WifiP2pDevice device = ...;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
p2pMgr.connect(channel, config, new ActionListener()
{
    @Override
    public void onSuccess()
    {
    }

    @Override
    public void onFailure(int reason)
    {
    }
});
```

Connexion

- WIFI_P2P_CONNECTION_CHANGED_ACTION est broadcasté lorsque la connexion est établie.
- Le reste de la communication (client-serveur) s’établit de manière classique, en utilisant des sockets.

```
@Override
public void onReceive(Context context, Intent intent)
{
    NetworkInfo info = (NetworkInfo) intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
    if (network.isConnected())
    {
        p2pMgr.requestConnectionInfo(channel, new WifiP2pManager.ConnectionInfoListener()
        {
            @Override
            public void onConnectionInfoAvailable(final WifiP2pInfo info)
            {
                InetAddress groupOwnerAddress = info.groupOwnerAddress.getHostAddress());
                if (info.groupFormed && info.isGroupOwner)
                {
                    // do whatever tasks are specific to the group owner (server)
                }
                else if (info.groupFormed)
                {
                    // the other devices acts as clients
                }
            }
        });

    }
}
```

SERVICE

- WiFi Direct autorise les protocoles de découverte de service classiques, comme Bonjour et UPnP, mais cette fois ci en environnement pair-à-pair.

```
@Override
public void onReceive(Context context, Intent intent)
{
    NetworkInfo info = (NetworkInfo) intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
    if (network.isConnected())
    {
        p2pMgr.requestConnectionInfo(channel, new WifiP2pManager.ConnectionInfoListener()
        {
            @Override
            public void onConnectionInfoAvailable(final WifiP2pInfo info)
            {
                InetAddress groupOwnerAddress = info.groupOwnerAddress.getHostAddress());
                if (info.groupFormed && info.isGroupOwner)
                {
                    // do whatever tasks are specific to the group owner (server)
                }
                else if (info.groupFormed)
                {
                    // the other devices acts as clients
                }
            }
        });

    }
}
```

NSD

- Network Service Discovery est une implémentation spécifique à Android de DNS-SD (DNS-based Service Discovery).
- Ce protocole permet notamment, une fois connecté à un réseau WiFi, de détecter l’ensemble des services exposés et de s’y connecter avec une simple socket.
  - Contrairement à WiFi Direct, donc, il est nécessaire d’être connecté à un point d’accès.

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

Exposer un service

- Un service est décrit par :
  - Un nom, que les autres appareils peuvent découvrir. En cas de conflit, Android ajoute un chiffre au nom de service.
  - Un type, qui précise le protocole de transport et le protocole applicatif, de la forme _<application>._<transport> (e.g., _http._tcp).
  - Un port (qui sera découvert par les autres appareils).

```java
NsdServiceInfo serviceInfo = new NsdServiceInfo();
serviceInfo.setServiceName("MyBeautifulInriaService");
serviceInfo.setServiceType("_http._tcp.");
serviceInfo.setPort(9000);

// get a random available port
ServerSocket server = new ServerSocket(0);
int port =  mServerSocket.getLocalPort();
```

Exposer un service

```
NsdManager.RegistrationListener rListener = new NsdManager.RegistrationListener()
{
    @Override
    public void onServiceRegistered(NsdServiceInfo NsdServiceInfo)
    {
        // the service name may have changed, if a conflict happened
        String serviceName = NsdServiceInfo.getServiceName();
    }

    @Override
    public void onRegistrationFailed(NsdServiceInfo serviceInfo, int errorCode)
    {
    }

    @Override
    public void onServiceUnregistered(NsdServiceInfo serviceInfo)
    {
    }

    @Override
    public void onUnregistrationFailed(NsdServiceInfo serviceInfo, int errorCode)
    {
    }
};

NsdManager nsdMgr = (NsdManager) getSystemService(Context.NSD_SERVICE);
nsdMgr.registerService(serviceInfo, NsdManager.PROTOCOL_DNS_SD, rListener);
```

Découvrir des services

```
final List<NsdServiceInfo> discoveredServices = new ArrayList<NsdServiceInfo>();
NsdManager.DiscoveryListener dListener = new NsdManager.DiscoveryListener()
{
      @Override
      public void onDiscoveryStarted(String serviceType)
      {
      }
      @Override
      public void onDiscoveryStopped(String serviceType)
      {
            // do things
      }

      @Override
      public void onServiceFound(NsdServiceInfo service)
      {
            if(service.getServiceName().equals("MyOwnService") == false && service.getServiceType().equals("_http._tcp"))
                  discoveredServices.add(service);
      }
      @Override
      public void onServiceLost(NsdServiceInfo service)
      {
            discoveredServices.remove(service); // the service is no longer available
      }

      @Override
      public void onStartDiscoveryFailed(String serviceType, int errorCode)
      {
            nsdMgr.stopServiceDiscovery(this);
      }
      @Override
      public void onStopDiscoveryFailed(String serviceType, int errorCode)
      {
            nsdMgr.stopServiceDiscovery(this);
      }
};

nsdMgr.discoverServices("_http._tcp", NsdManager.PROTOCOL_DNS_SD, dListener);
```

Connexion à un service

- Lorsqu'un service est découvert, la première étape consiste à rapatrier les informations nécessaires à la connexion.

```
NsdManager.ResolveListener sListener = new NsdManager.ResolveListener()
{
    @Override
    public void onResolveFailed(NsdServiceInfo serviceInfo, int errorCode)
    {
    }

    @Override
    public void onServiceResolved(NsdServiceInfo serviceInfo)
    {
        if (serviceInfo.getServiceName().equals("MyOwnService")) // it's my own service !
            return;


        int port = serviceInfo.getPort();
        InetAddress host = serviceInfo.getHost();

        ...
    };
}

nsdMgr.resolveService(discoveredServices.get(0), sListener);
```

Connexion à un service

- Une fois la résolution effectuée, il ne reste plus qu’à utiliser des sockets classiques.
- Ne pas oublier de stopper la découverte et de supprimer le service lors de la fermeture de l’application.

```
@Override
protected void onDestroy()
{
    nsdMgr.unregisterService(rListener);
    nsdMgr.stopServiceDiscovery(dListener);
    super.onDestroy();
}
```

NFC