Développer son application Android

13 avril 2012 Daniel Bort

La semaine dernière nous vous proposions la mise à jour de notre tuto d'installation de l'environnement de développement Android ! Ce tuto était le premier d'une nouvelle série de tutos où nous allons nous mettre pour de bon au dév Android !

tuto développement Android pour les noobs

En développement Android je suis un gros débutant, sûrement comme la plupart d'entre vous.
Mais voilà, coder mes applications pour Android me tente assez.
Je vais donc partager avec vous mes expériences, mes avancées à travers des **tutos simples**, clairs, illustrés et surtout accessibles !

Les premiers tutos seront hyper basiques mais nous y aborderons des notions essentielles, les bases. Certains raleront car les applications réalisées au cours de ces TP seront laides et basiques mais pour le look et les fonctionnalités avancées nous verrons plus tard. Ici on apprend et on part de zéro !

Dans ce premier tuto nous allons gérer le click d'un bouton, récupérer le texte saisi dans un champs texte, l'afficher et traiter le texte saisie (si c'est une URL l'ouvrir dans notre première application). Pour le fun j'ai aussi ajouté une barre de progression que nous allons gérer très simplement.

En terme plus professionnels : nous allons poser des écouteurs d'évènements (**Listener**), intercepter des évènements comme le **onClick** d'un **Button**, le **onPageFinished** d'un **WebView**, lire le texte saisi dans un **EditText** et vérifier son contenu, afficher ce texte dans un **TextView**, naviguer sur internet grâce à un **WebView**, gérer deux états d'une **ProgressBar**. Et tout ça dans 2 **Layout** ! Ça fait pas mal de mots barbares déjà 😉

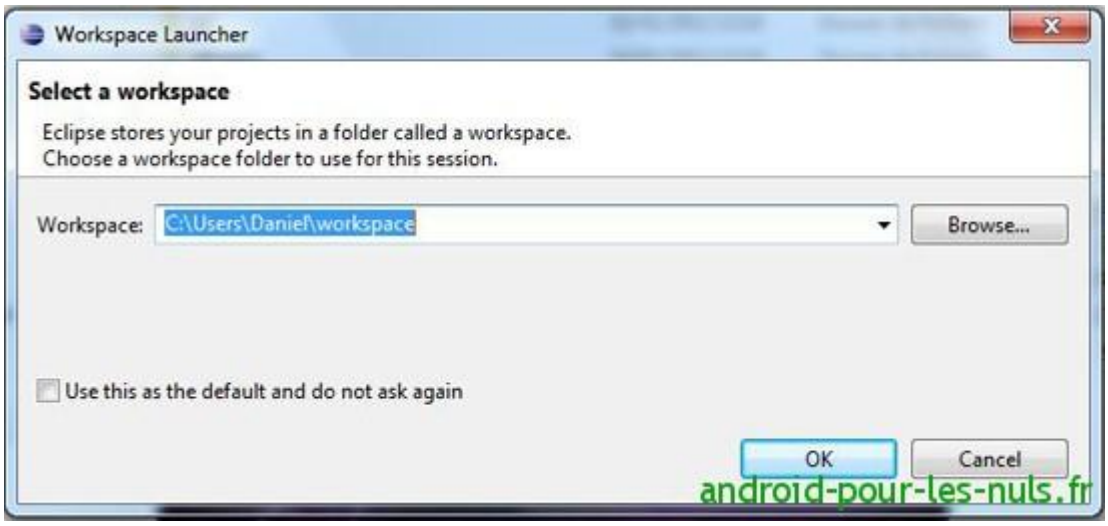
Bon assez de blah blah non ? Allez on attaque !

1 – Création du projet Eclipse

Si vous ne l'avez pas encore fait, installez l'environnement de développement Eclipse – SDK Android – ADT.
Si vous avez bien suivi ce tuto votre **Workspace** (espace de travail) est déjà défini dans

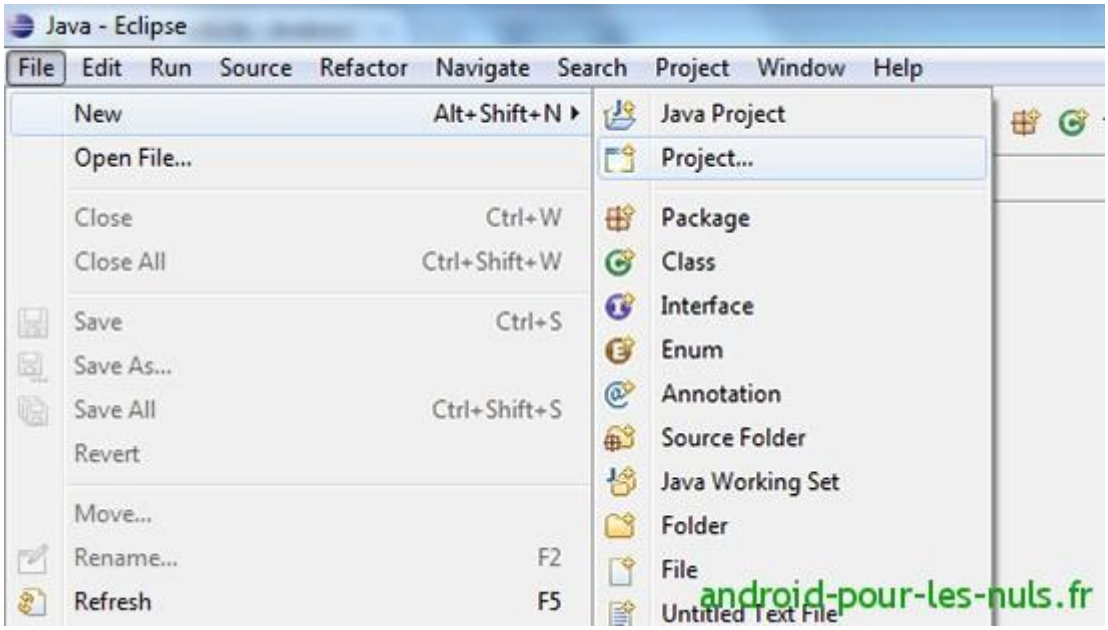
« C:\Eclipse\Workspace » sinon faites le ; Eclipse au lancement vous demande où se trouve son Workspace et au besoin vous pourrez le re-spécifier.

Lancez Eclipse aussi : double click sur « Eclipse.exe » 😀

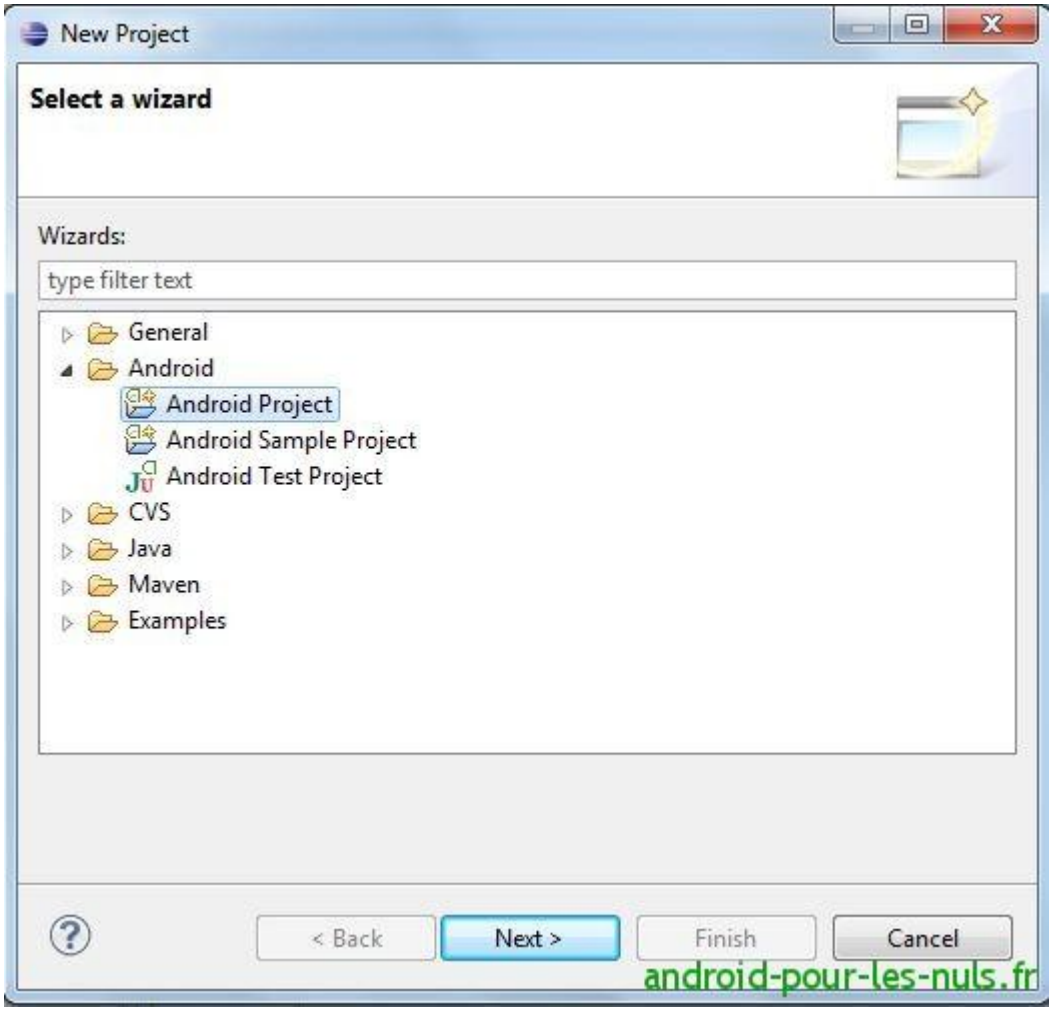


Nous allons créer un nouveau projet dans notre Workspace.

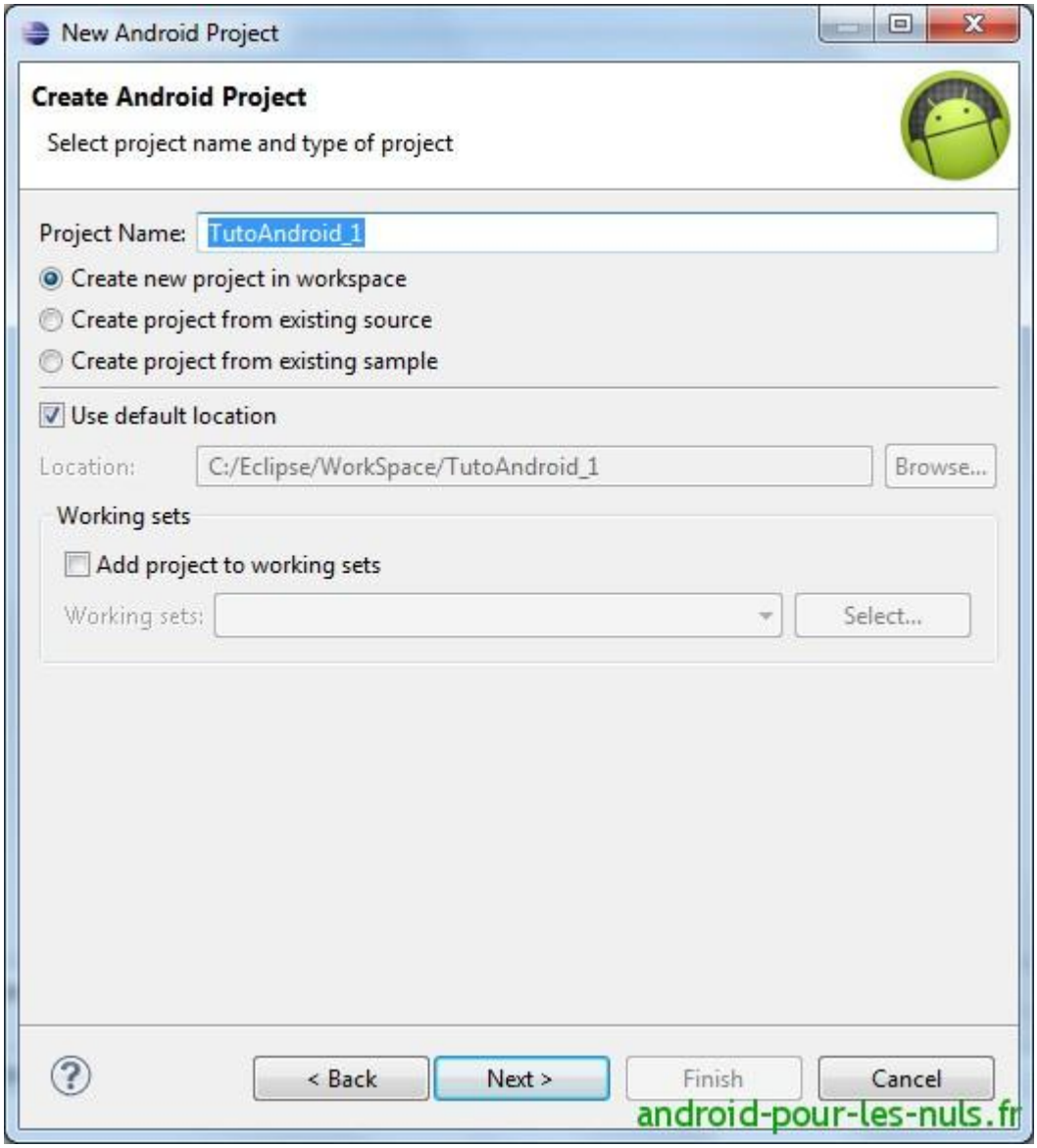
Dans Eclipse : File / News / Project



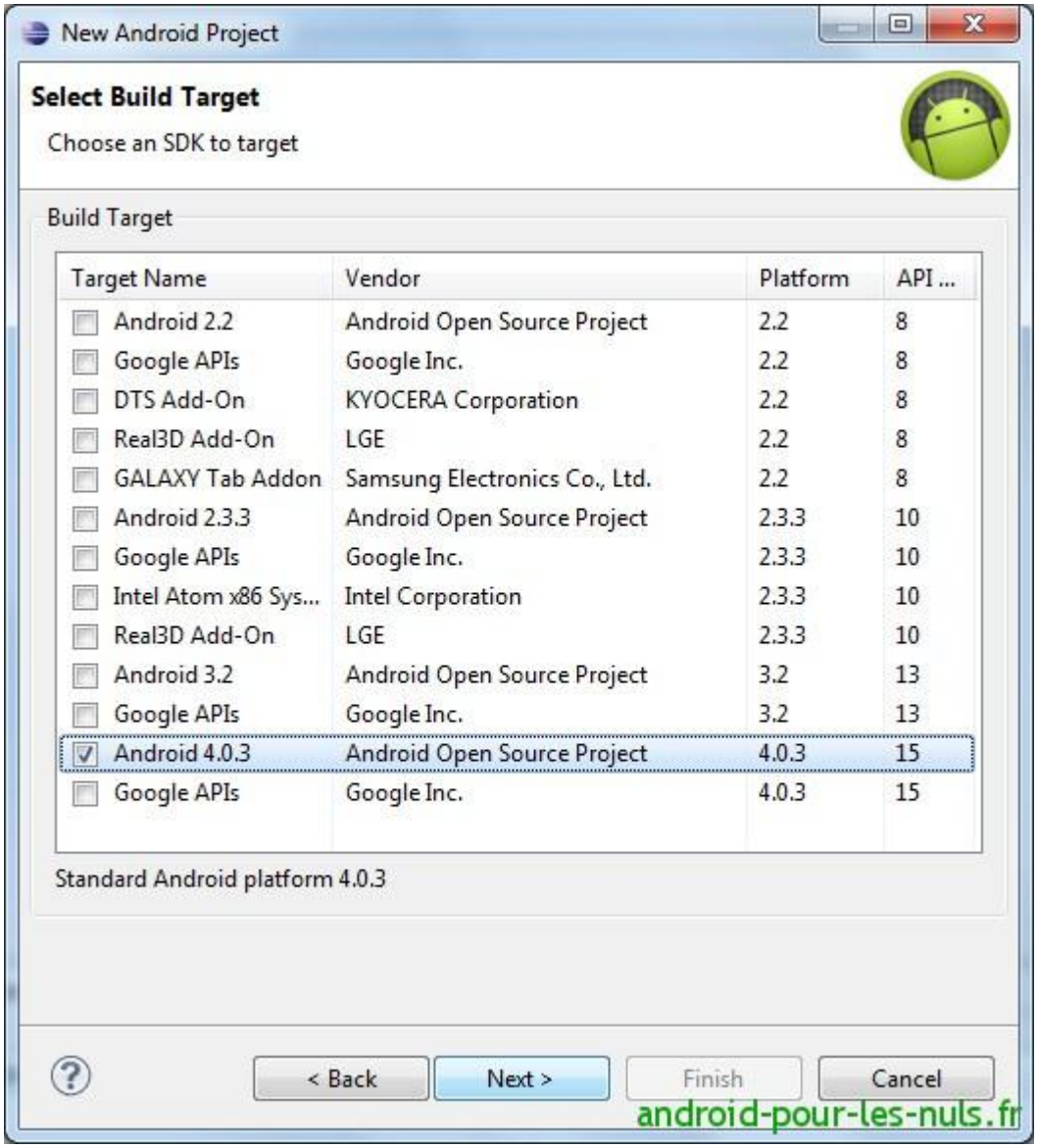
Puis Android Project



Ici nous allons nommé notre projet « TutoAndroid_1″

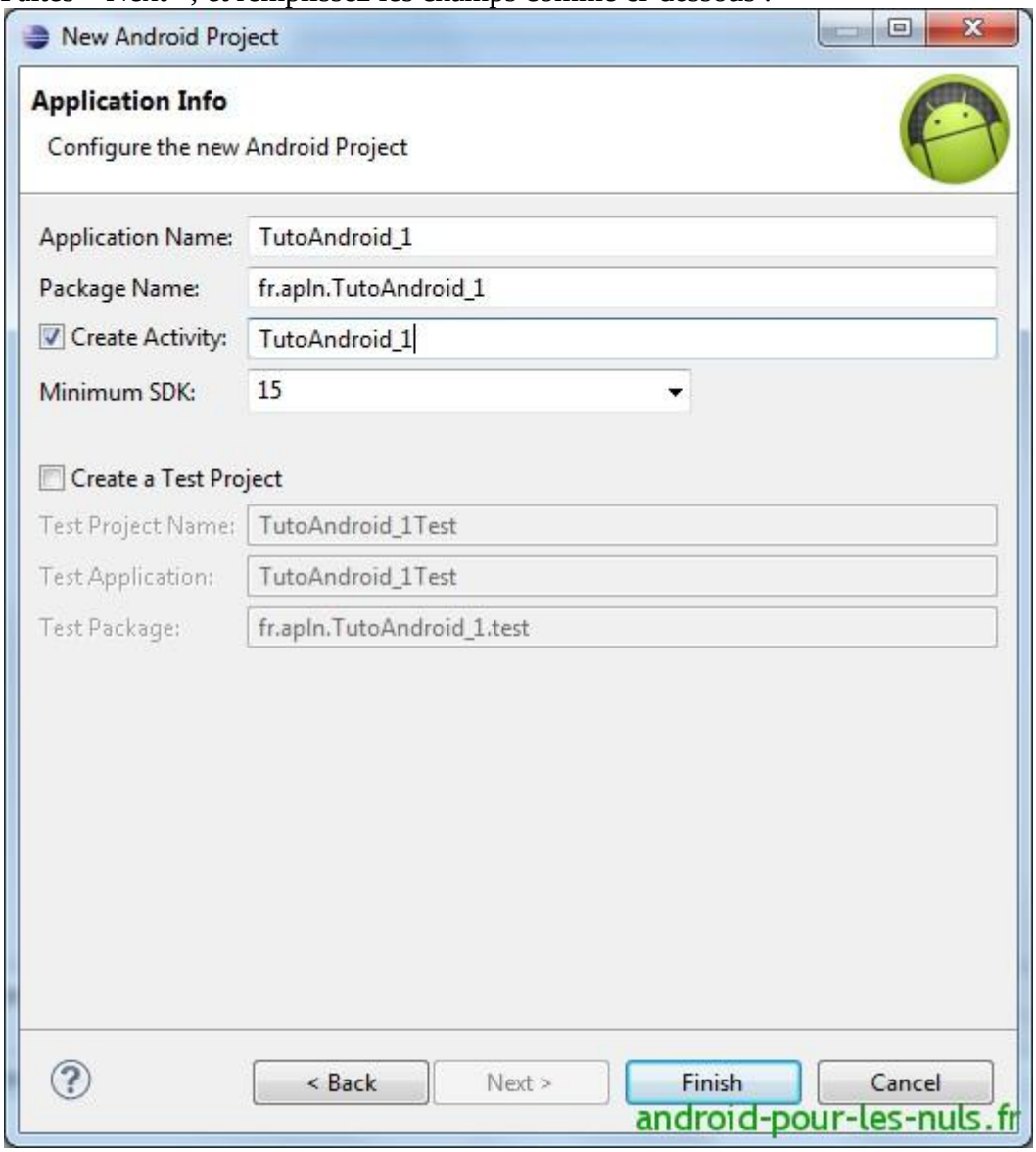


puis « Next ».
Puis nous allons choisir la version mini d'Android pour notre application.
Nous sommes à l'époque de Ice Cream Sandwich, choisissons Android 4.03 :

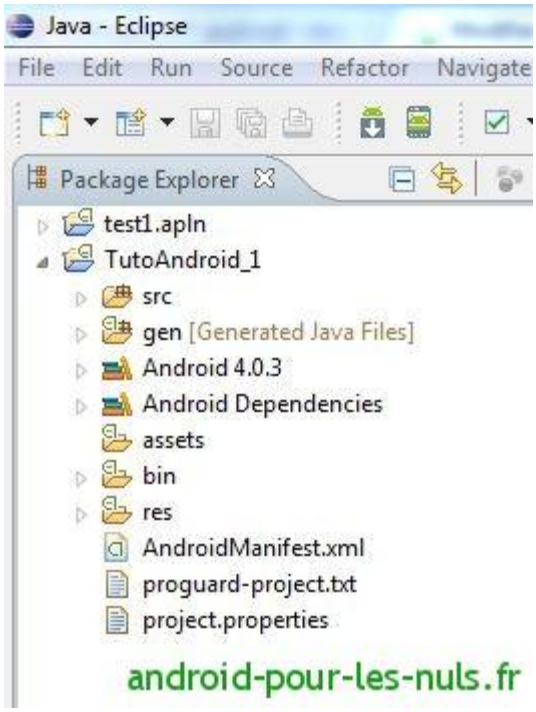


Si besoin, pour faire tourner l'application sous Android 2.3 Gingerbread nous pourrons le changer dans les propriétés du projet et ceci même à la fin quand l'application tournera dans l'émulateur de ICS.

Faites « Next », et remplissez les champs comme ci-dessous :

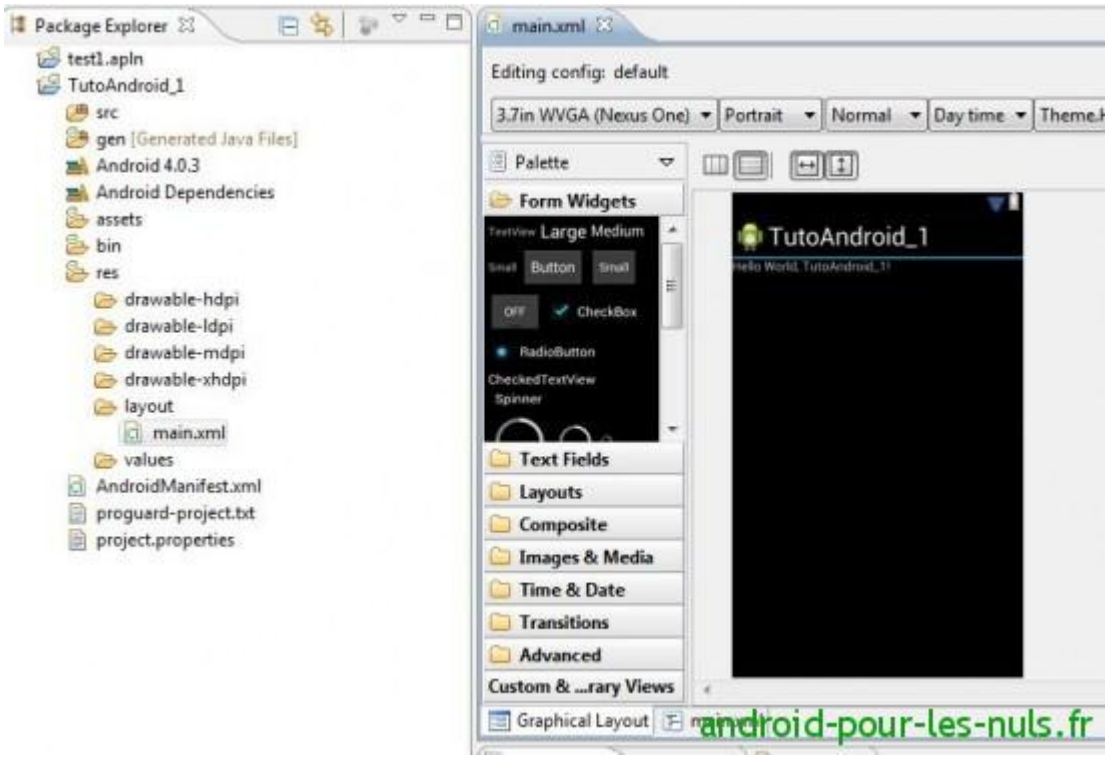


et « Finish ».
Dans les prochains tutos nous passerons plus vite sur ces étapes.
Dans le **Package Explorer** d'Eclipse nous avons maintenant notre nouveau projet :



2 – Les fichiers de notre projet Eclipse

Notre application ne contiendra qu'une seule page (nommée **Activity** dans le language Android), cette page est représentée par le fichier xml de définition de l'interface : « main.xml » qui se trouve dans le dossier « res/layout » de notre projet.

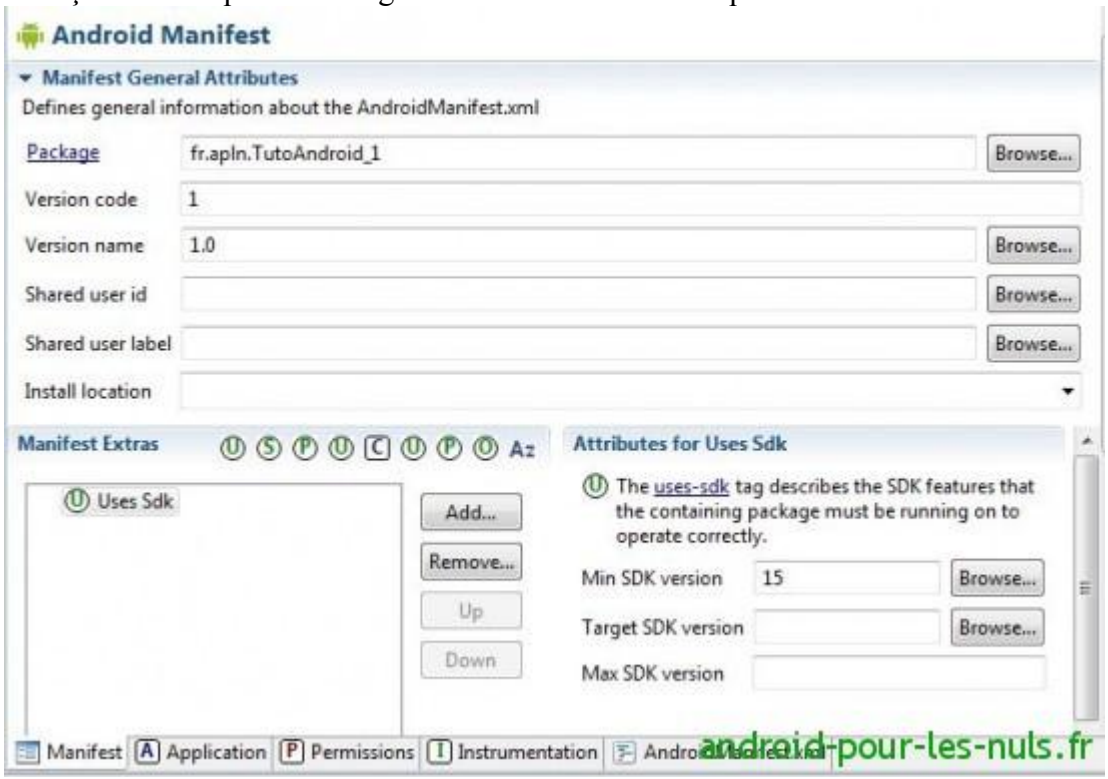


Le code source de notre application est lui contenu dans le fichier »TutoAndroid_1.java » qui se trouve dans le dossier « src » et sous « fr.apln.TutoAndroid_1″ dans l'arborescence de notre premier tuto.

Un autre fichier « remarquable » que nous allons modifier est « AndroidManifest.xml » qui lui est à la racine du projet :

L'affichage dans la fenêtre d'édition (à gauche du Package Explorer) est une aide à l'edition du »AndroidManifest.xml ».
Nous allons le modifier directement dans son code source.

Pour ça on va cliquer sur l'onglet AndroidManifest.xml qui se trouve en bas de la fenêtre :



Le source xml du fichier donne :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.apln.TutoAndroid_1"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".TutoAndroid_1"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

On y retrouve la version mini d'android pour faire tourner notre appplication.
Ici Ice Cream Sandwich 4.0.3 qui correspond à la version 15 du SDK. Le nom de notre package : »fr.apln.TutoAndroid_1″.

Dans le **Manifeste** on va rajouter un droit à l'application.
En effet notre application pourra accéder à Internet. Nous allons donc ajouter la propriété « android.permission.INTERNET » dans une balise « uses-permission ».

Voici quelques exemples de permissions que l'on peut ajouter à une application :

android.permission.CALL_EMERGENCY_NUMBERS
android.permission.READ_OWNER_DATA

android.permission.SET_WALLPAPER
android.permission.DEVICE_POWER

Il y en a plein d'autres comme lancer un appel, envoyer des SMS …

Nous allons ajouter la ligne suivante à notre manifeste

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

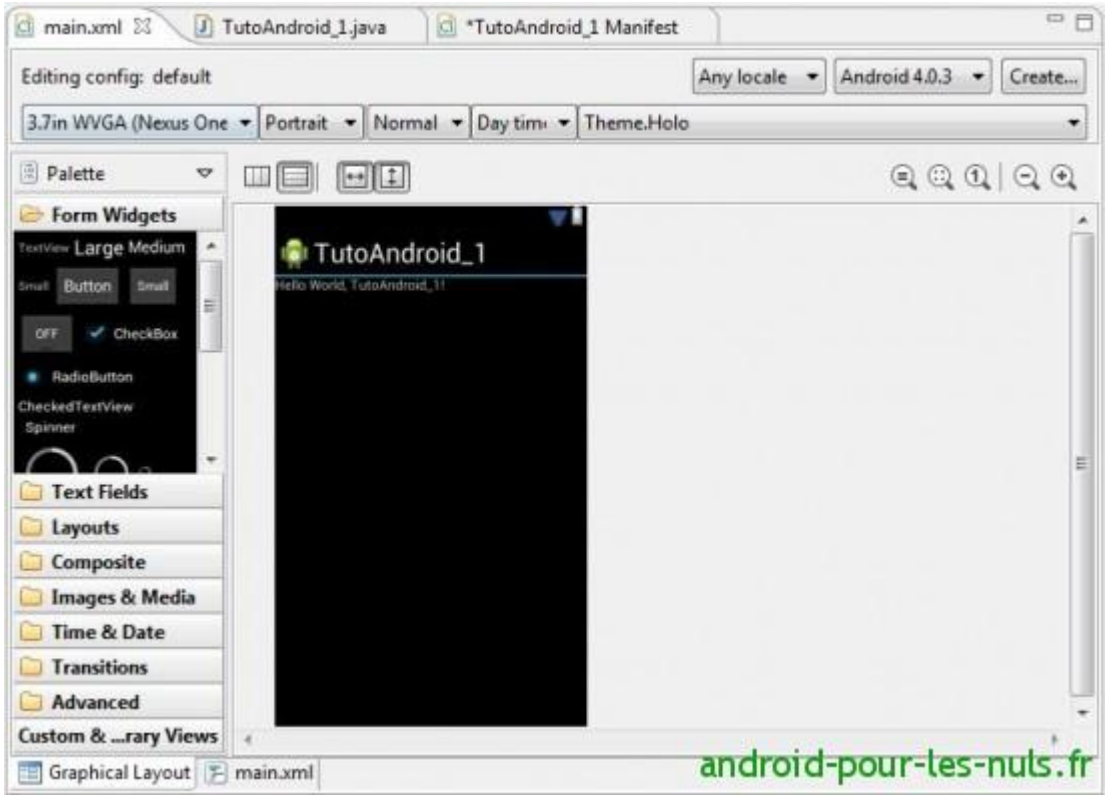
qui devient :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.apln.TutoAndroid_1"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".TutoAndroid_1"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

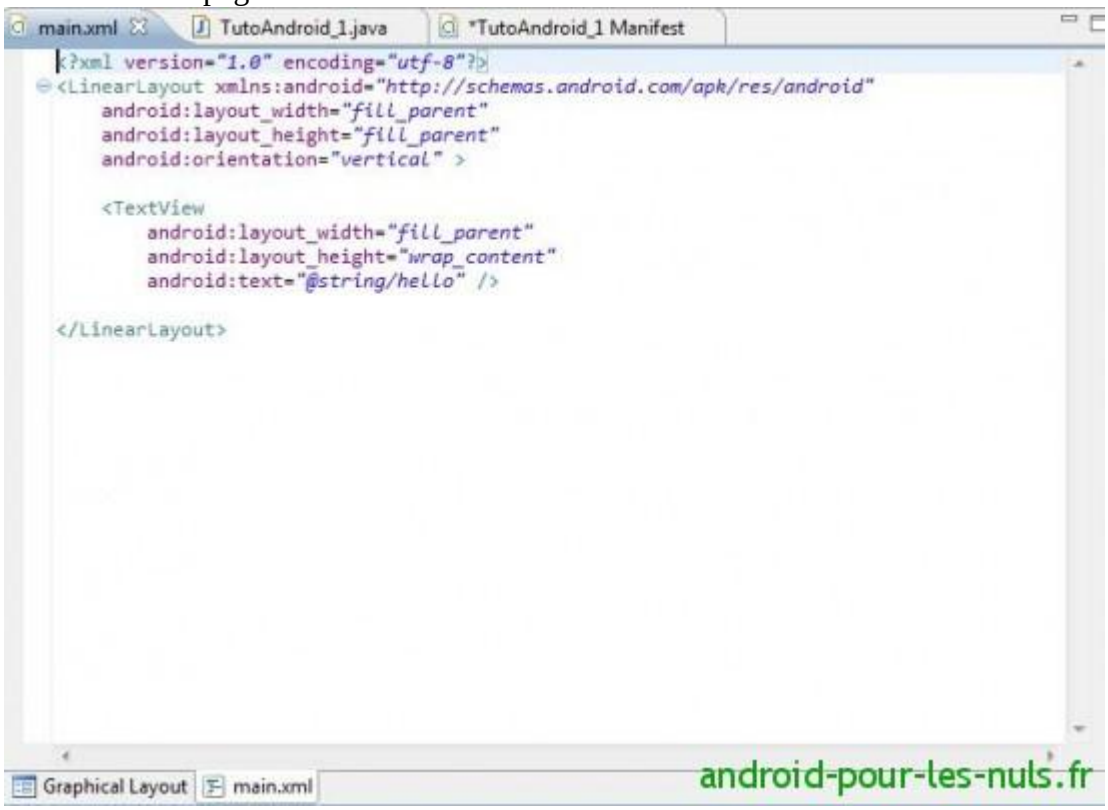
Sauver par la combinaison de touche Ctrl+S ou avec l'icone « Save » qui se trouve dans la barre d'icones de Eclipse (la disquette bleue).
Ces 3 fichiers présentés ici seront pour l'instant les seuls que nous allons modifier.
Nous verrons les autres fichiers et dossier dans nos prochains tutos.

3 – Construction de l'interface de notre application

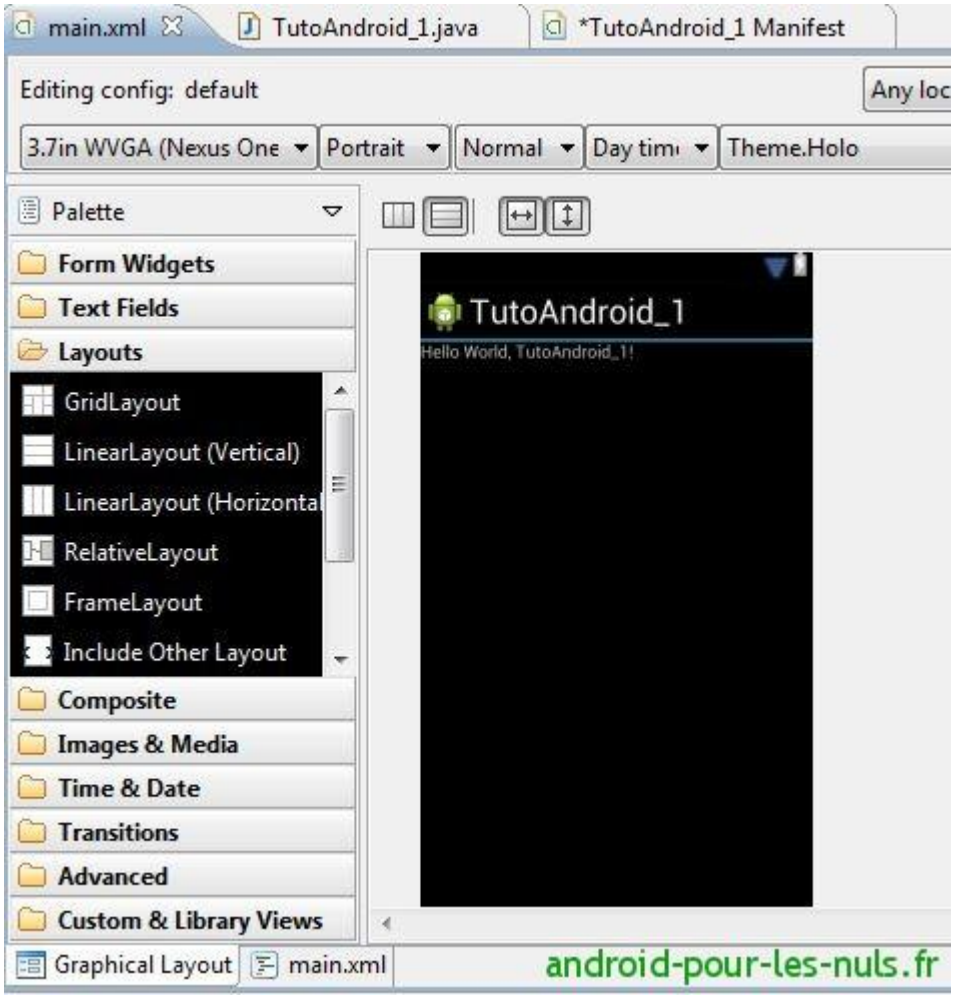
Notre interface, la page de notre application, est définie dans le fichier « main.xml ».
Eclipse nous propose un éditeur ou l'on pourra construire l'interface par « drag and drop » :



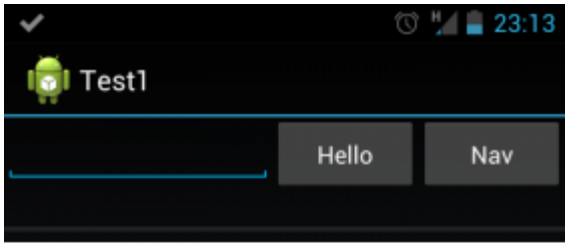
Nous verrons que le drag and drop montrera très vite ses limites et nous serons contraints de mettre les mains dans le cambouis xml du code source de notre page qui rappelez vous est aussi appelée **Activity :**
En cliquant du l'onglet « main.xml » en bas de cette fenêtre d'édition , nous ouvrons l'éditeur de code source de notre page :



Pour l'instant notre page est très simple. Elle comporte un **LinearLayout** en orientation verticale.
Ce LinearLayout est un **container** où tous les éléments graphiques de notre interface s'empilent les uns au dessus des autres automatiquement.
Il existe différentes formes de Layout, pour les voir cliquer sur l'onglet « Graphical Layout » pour retourner dans l'éditeur graphique de l'activity puis déployer « Layouts » dans la partie gauche de la fenêtre :



Ici nous n'utiliserons que le LinearLayout en mode horizontal (pour empiler sur le côté gauche) et en mode vertical (pour empiler vers le bas).
Notre application (moche ok) doit ressembler à ça :

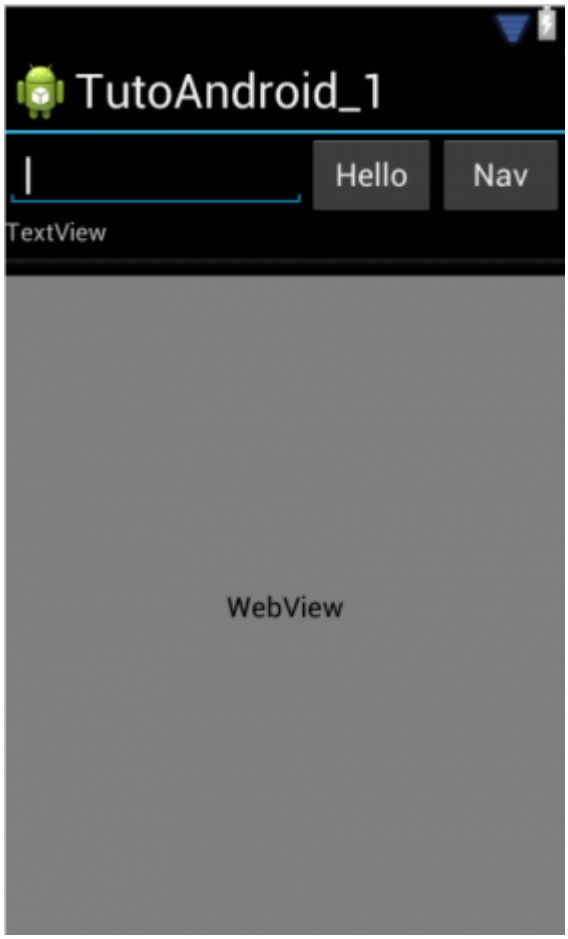


En haut un champ texte de saisie : un **EditText**
Et à sa gauche un **Button** « Hello » suivi d'un autre **Button** « Nav ».En dessous du **WebView** (un navigateur web) on a un champ texte non éditable, un **TextView** et juste en dessous de ce textview on a une **ProgressBar.**
Schématiquement nous aurons :

Un Layout en mode vertical principal en orange saumon, un Layout en mode horizontal en bleu qui contient 3 éléments puis encore en dessous 3 autres éléments.
Le code source de notre « main.xml » sera :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <EditText
            android:id="@+id/editText1"
            android:layout_width="172dp"
            android:layout_height="wrap_content"
            android:ems="10" >

            <requestFocus />
        </EditText>

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Hello" />

          <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Nav" />

    </LinearLayout>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:indeterminate="false" />

    <WebView
        android:id="@+id/webView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Ce qui nous donne en mode Graphical Layout :



Nous ne gérerons par ici la notion de rotation de l'application, nous verrons ça dans des tutos à venir.

Attardons nous sur le Layout en mode horizontal avec son EditText et ses 2 Buttons.
Comme toute balise XML elle s'ouvre, se ferme et possède des attributs
<LinearLayout ses attributs>…… </LinearLayout>
et pour ses attributs :

android:layout_width= »match_parent » android:layout_height= »wrap_content » qui veulent dire que le Layout prendra toute la place disponible verticalement et horizontalement.

Dans ce Layout :

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="172dp"
    android:layout_height="wrap_content"
    android:ems="10" >
    <requestFocus />
</EditText>
```

Le champ texte de saisie :

nous allons voir comment « récupérer » le texte saisi dans le code source java de l'application.

Et les boutons :

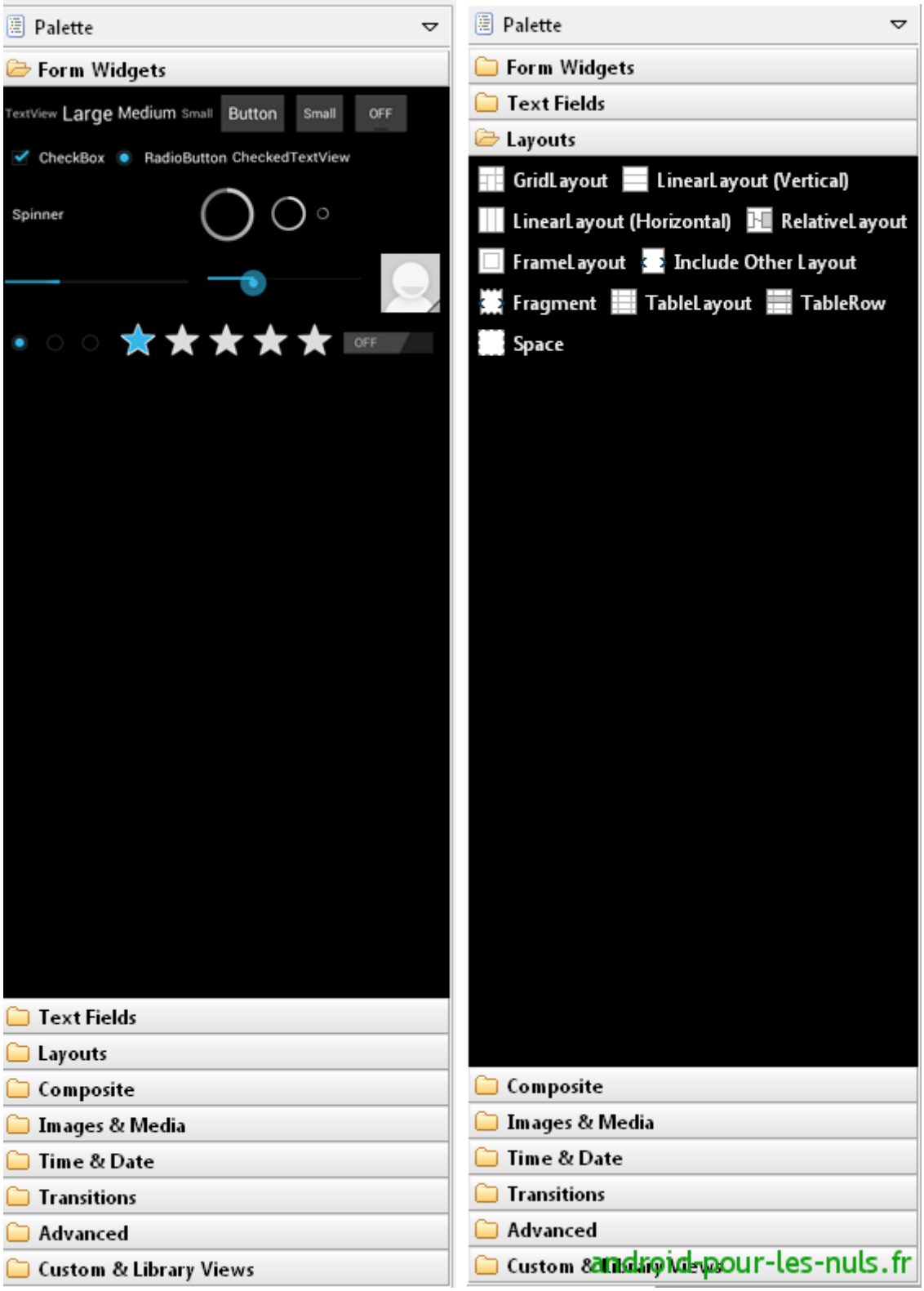
pour les boutons nous allons intercepter l'évenement « click » qui déclenchera une action.

Le texte qui s'affiche sur les boutons et leur nom (qui doit être unique) sont fixés dans leurs déclarations en xml :

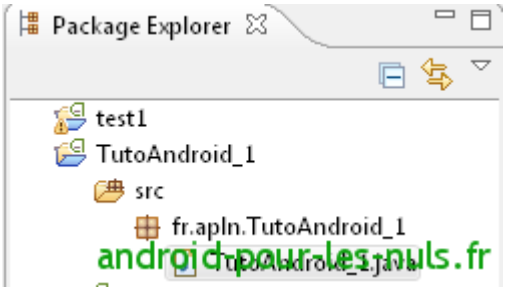
```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Hello" />

  <Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Nav" />
```

Le premier bouton s'appelle « button1″ et son texte est fixé a « Hello ».
Nous avons construit l'interface de notre première application à la main en XML mais nous pouvons aussi la construire via **Drag en Drop** en mode Graphical Layout.

Pour avoir un aperçu de tous les éléments dont nous disposons, ouvrez tous les menus de la Palette :



Il faudra commencer par déposer les Layout sachant que par défaut l'application contient déjà un Layout en mode vertical.
Amusez vous avec ce mode de construction d'interface, vous découvrirez vite ses limites et devrez éditer à la main le code source du main.xml afin de placer vos éléments exactement où vous les voulez.
Jetons un oeil maintenant au **code Java** de l'application car pour l'instant notre application ne sait rien faire.
Le code source de l'application se trouve ici :



Et pour l'instant il n'y a que ça :

```
package fr.apln.TutoAndroid_1;

import android.app.Activity;
import android.os.Bundle;

public class TutoAndroid_1 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

On retrouve le **package** que nous avions définis lors de la création du projet : « package fr.apln.TutoAndroid_1″

Puis les imports qui représente les éléments (d'autres packages) que l'application utilisera, nous devrons en rajouter.

Puis la déclaration de notre **classe java** qui hérite d'Activity. On se prend pas la tête avec ses notion pour l'instant.

Ensuite nous avons »public void onCreate(… » … mais c'est quoi ? C'est le code qui est exécuté lors du lancement de l'application (on va dire comme ça pour faire simple).

On remarque que pour l'instant il n' y a aucune trace de nos boutons, champs textes, barre de progression que nous avons mis dans notre « main.xml ».
Ne vous inquiétez pas nous allons spécificier ces éléments, les nommer et dire ce qu'ils doivent faire.

3 – Code source java de notre application Android

Commençons par notre EditText (le champ de saisie) qui s'appelle »editText1″ dans le code source XML, nous allons le déclarer dans le code source :
En dessus de

```
setContentView(R.layout.main);
```

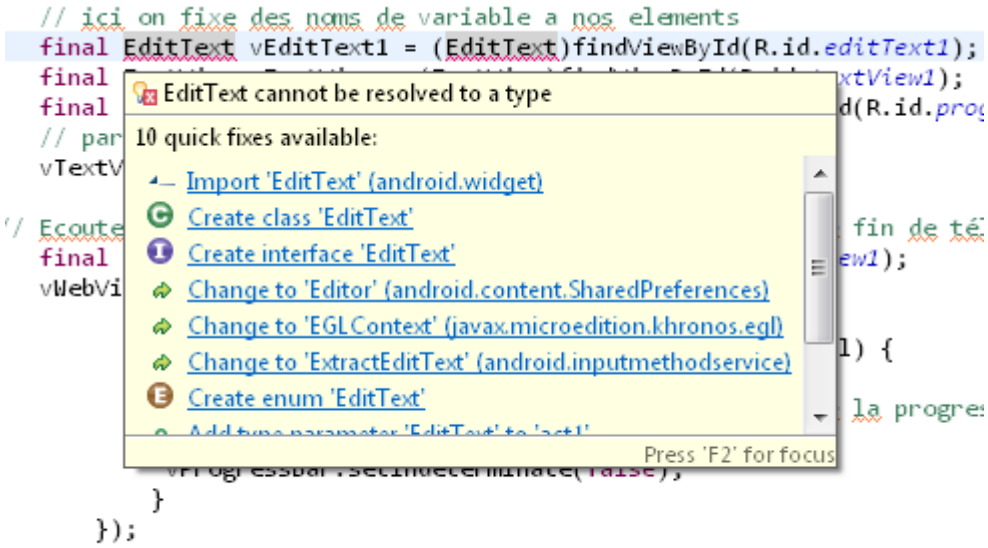
Ajoutez

```
 final EditText vEditText1 = (EditText)findViewById(R.id.editText1);
```

En clair, on crée une variable nommé vEditText1 de type EditText qui est associé à l'élément d'interface nommé editText1.
Voici ce que nous avons dans notre code source :

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // ici on fixe des noms de variable a nos elements
    final EditText vEditText1 = (EditText)findViewById(R.id.editText1);
```


EditText est souligné en rouge, cela veut dire que notre application ne connait pas le type « EditText ». Mettez le curseur de la souris sur EditText :



cliquez sur le lien « Import « EditText » … ce qui aura pour effet d'ajouter une ligne dans nos imports :

```
import android.widget.EditText;
```

Maintenant que nous avons vu la procédure pour un import nous pouvons ajouter tous les imports des packages que nous allons utiliser :

```
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;
```

Déclarons maintenant le TextView et la ProgressBar :

```
final TextView vTextView = (TextView)findViewById(R.id.textView1);
final ProgressBar vProgressBar = (ProgressBar)findViewById(R.id.progressBar1);
```

Maintenant passons aux élements qui vont entraîner des actions dans notre application. Le Bouton « Hello » par exemple :

```
        Button vButton1 = (Button)findViewById(R.id.button1);
        vButton1.setOnClickListener(new View.OnClickListener()
        {
                public void onClick(View v)
                {

                }
        });
```

Dans la première ligne, on déclare une variable nommée vButton1 de type Button représentée par l'élément « button1″ dans le xml.
La deuxième ligne est plus intéressante car on commence à parler **d'évènements**.

On place un **Listener** sur le Bouton, c'est à dire que le bouton se met à l'écoute des évènements de « OnClick » et l'on déclare une méthode Onclick qui sera exécutée lors d'un click sur le bouton « Hello ». Nous placerons le code de ce que nous voulons que l'application fasse entre les « {} » de la méthode :

```
public void onClick(View v)
{
   // code qui sera executer lors d'un click sur le bouton Hello

}
```

Déclarons le WebView :

```
final WebView vWebView = (WebView)findViewById(R.id.webView1);
vWebView.setWebViewClient(new WebViewClient() {

          public void onPageFinished(WebView view, String url) {

                // quand la page a fini de s'afficher on repasse la progress bar à
l'etat Indeterminate = false
                // cela mettra fin à son animation
                vProgressBar.setIndeterminate(false);
            }
        });
```

La WebView est mise à l'écoute des évènement du web. Quand elle recoit un évènement de fin de chargement de page »onPageFinished » le méthode de même nom s'exécutera.

Dans cette méthode on fixera l'état de la **ProgressBar** à un état déterminé ce qui aura pour effet d'en arrêter l'animation. En effet lorsque l'on fixe une progressBar à l'état indéterminé elle s'anime. Vous le verrez lors du premier lancement de l'application dans l'émulateur.
Maintenant déclarons l'autre bouton :

```
Button vButtonWeb = (Button)findViewById(R.id.button2);
vButtonWeb.setOnClickListener(new View.OnClickListener()
{
        public void onClick(View v)
        {
                vProgressBar.setIndeterminate(true);
                vTextView.setText("Go go go !!");
                vWebView.loadUrl("http://www.android-pour-les-nuls.fr");

        }
});
```

Un click sur ce bouton aura pour conséquence de :
Passer la progressBar à l'état indéterminé (pour qu'elle s'anime), afficher « Go go go !! » dans le TextView et ouvrir la page de notre site dans le WebView.

Maintenant nous allons coder l'action qui sera faite après un click sur le bouton « Hello ». voici ce que nous allons coder mais en français :

- Récupérer le texte saisi dans le EditText
si
le texte saisi est une url genre « google.fr » donc contient un « . »

alors
lancer l'animation de la progressbar
ajouter « http://www. » au texte saisie afin d'avoir une URL complète
ouvrir cette url dans la webView
afficher « Go to » et le texte saisi dans le TextView
sinon
afficher « Hello » et le texte saisi dans le TextView

Ce qui donne en Java pour la méthode onClick :

```
public void onClick(View v)
{
    String vTexteSaisie = vEditText1.getText().toString();
    // si le texte saisie contient .
    if (vTexteSaisie.contains("."))
    {
       // on transforme de texte saisi en URL
       // ex : on saisit "free.fr" on naviguera vers "http://www.free.fr"
       // on force la progressbar a Indeterminate = true comme ca elle est animée
       vProgressBar.setIndeterminate(true);
       vWebView.loadUrl("http://www."+vTexteSaisie);
       vTextView.setText("Go to "+vEditText1.getText()+" !!");
    }
    // sinon
    else
       vTextView.setText("hello "+vEditText1.getText()+" !!");
}
```

Donc le code source de l'application donne :

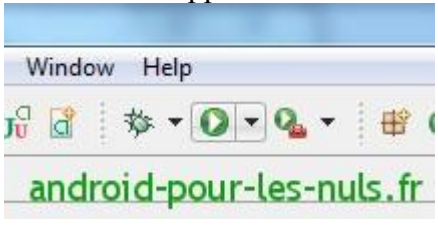
```
package fr.apln.TutoAndroid_1;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;

public class TutoAndroid_1 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // ici on fixe des noms de variable a nos elements
        final EditText vEditText1 = (EditText)findViewById(R.id.editText1);
        final TextView vTextView = (TextView)findViewById(R.id.textView1);
        final ProgressBar vProgressBar = (ProgressBar)findViewById(R.id.progressBar1);
        // par code on efface de texte du TextVeiw
        vTextView.setText("");

     // Ecouteur sur le WebView. on intercepte ici l'evenement de fin de téléchargement
de la page
        final WebView vWebView = (WebView)findViewById(R.id.webView1);
        vWebView.setWebViewClient(new WebViewClient() {

                    public void onPageFinished(WebView view, String url) {
```

```
                        // quand la page a fini de s'afficher on repasse la progress
bar à l'etat Indeterminate = false
                        // cela mettra fin à son animation
                        vProgressBar.setIndeterminate(false);
                    }
                });

        // Ecouteur sur le bouton Nav
        Button vButtonWeb = (Button)findViewById(R.id.button2);
        vButtonWeb.setOnClickListener(new View.OnClickListener()
        {
                public void onClick(View v)
                {
                        vProgressBar.setIndeterminate(true);
                        vTextView.setText("Go go go !!");
                        vWebView.loadUrl("http://www.android-pour-les-nuls.fr");

                }
        });

     // Ecouteur sur le bouton Hello
        Button vButton1 = (Button)findViewById(R.id.button1);
        vButton1.setOnClickListener(new View.OnClickListener()
        {
                public void onClick(View v)
                {

                        String vTexteSaisie = vEditText1.getText().toString();

                        // si le texte saisie contient .
                        if (vTexteSaisie.contains("."))
                        {
                                // on transforme de texte saisi en URL
                                // ex : on saisit "free.fr" on naviguera vers
"http://www.free.fr"

                                // on force la progressbar a Indeterminate = true comme
ca elle est animée
                                vProgressBar.setIndeterminate(true);
                                vWebView.loadUrl("http://www."+vTexteSaisie);
                                vTextView.setText("Go to "+vEditText1.getText()+" !!");

                        }
                        // sinon
                        else
                                vTextView.setText("hello "+vEditText1.getText()+" !!");
                }
        });

    }
}
```

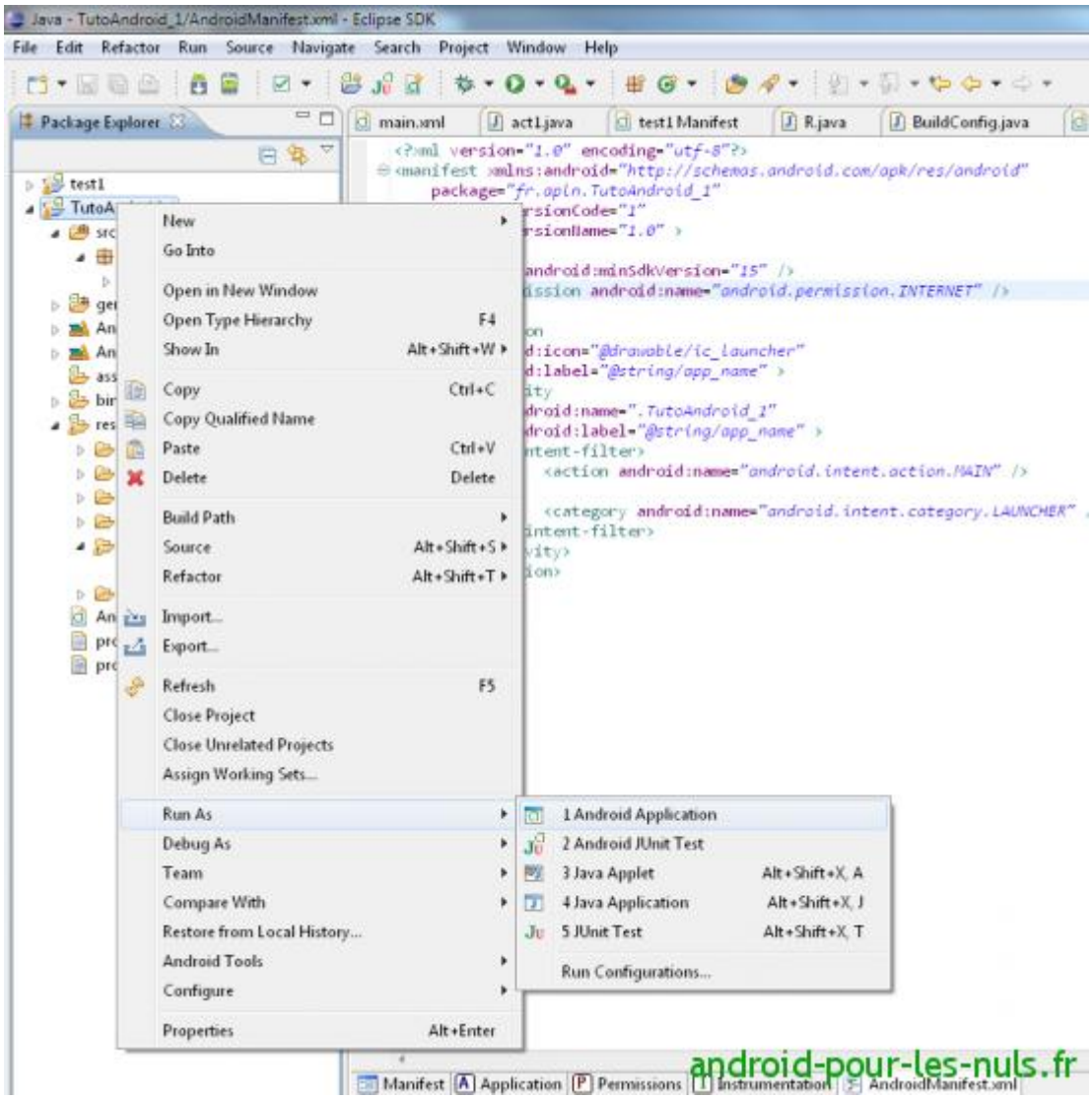
Les lignes commençant par « // » sont des lignes de commentaires, elles aident à la compréhension du code et dans des applications plus complexes à la maintenance en indiquant ce que le développeurs a voulu coder.

4 - Exécution de notre première application Android

Pour lancer l'application dans l'émulateur cliquez sur la flèche verte :

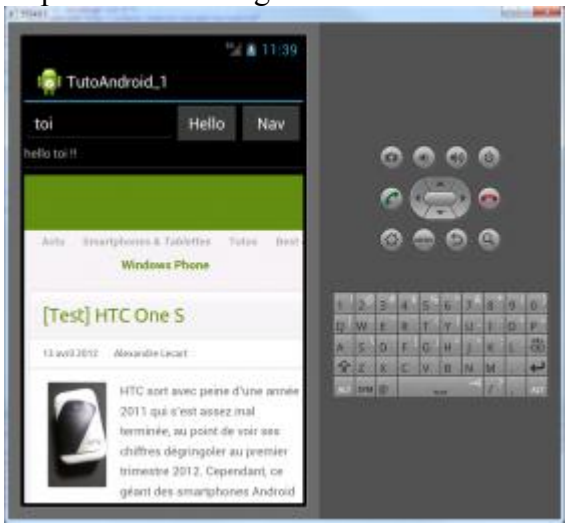


Si vous avez plusieurs projets dans votre workspace pour en lancer un spécifiquement : faites click droit sur votre projet puis « Run As / Android Application » :



L'AVD ICS se lance (si vous n'avez pas l'AVD (téléphone virtuel) en Android 4.0.3 ICS) reportez vous au tuto d'installation de l'environnement de Dev Android.

Le premier démarrage de l'émulateur est assez long, ICS démarre et notre application se lance.



voici la vidéo de l’application dans l’émulateur :

Nous avons donc notre **première application pour Android** codée par nos soins, elle est certes sommaire mais c’est la nôtre 😀

Si vous avez des questions, des remarques, des idées sur ce tuto ou ceux à venir, n’hésitez pas à nous en faire part dans les commentaires, nous lisons tout même si nous n’avons pas forcément le temps de répondre à tous 😉

Allez maintenant à vos Eclipse et à vos claviers ! 😀