This presentation is based on Jean-Paul Jamont’s one (Université Pierre Mendès France, IUT de Valence)
Agenda

Android Overview

Basics

Android Projects

Activities

XML Layout

Manifest

Some Android-specific Classes

References
Agenda

Android Overview
- What is Android?
- Who is Using Android?
- The Challenges
- Downloading and Uploading Apps
- Architecture logicielle

Basics

Android Projects

Activities

XML Layout

Manifest

Some Android-specific Classes

References
What is Android?

An open source operating system

- A mobile-device-oriented operating system
  - It consists in a set of software to interface hardware and applicative software

- An open source operating system
  - source code is available
  - huge developper community

- An operating system based on the Linux kernel

- A free development environment
  - Programming with Java or C language
A lot of versions...
The different versions of Android are all named as desserts since version 1.5 and follow an alphabetical order:

- 1.0 – 2007: little known version (distributed before the first Android phone)
- 1.1 – 2008: version included in the first Android phone, HTC Dream
- 1.5 - Cupcake - 04/2009: last official revision in May 2010
- 1.6 - Donut - 09/2009: last official revision in May 2010
- 2.0 - Eclair - 2009: quickly replaced by versions 2.0.1 then 2.1 due to major bugs
- 2.1 - Eclair - 01/2010: last official revision in May 2010
- 2.2 (2.2.3) - FroYo - 05/2010: last official revision in 2011
- 2.3 (2.3.7) - Gingerbread - 12/2010: version for smartphones and small tablets
- 3.0 (3.2) - Honeycomb - 01/2011: version for large tablets and TVs
- 4.0 (4.0.3) - Ice Cream Sandwich - 10/2011: unified version for smartphone, tablet and GoogleTV
- 4.1 (to 4.3) - Jelly Bean - 09/2012: smoother and more responsive feel
- 4.4 - KitKat
- 5.0 - Lollipop
- 6.0 - Marshmallow - current version
Android Platforms: More and more equipments

Phones and PDA

Tablets

Google TV

Consoles
Android and its competitors

Open Handset Alliance

- **Creation**: 5th November 2007, initiated by Google
- **Objectives**: Develop open norms for mobile phone devices
- **Members**: 84 technology and mobile companies

Competitors

- Apple with iOS
- Research In Motion (RIM) with BlackBerry OS
- Microsoft with Windows Phone
- Samsung with Bada (even Samsung is more and more using Android)
- HP with Palm webOS then webOS (stopped in 2011)
- Nokia with Symbian OS (stopped in 2011, Nokia is using now Windows Phone)
- ...
Phone Sales by OS

Share of worldwide 2013 Q2 smartphone sales to end users by operating system, according to Gartner
The Challenges

Android Functionnalities

- Applicative Framework
- Dalvik Virtual Machine
- Integrated Web Browser
- 2D and 3D Graphical Library
- SQLite
- Audio and Video Codecs
- WiFi, EDGE, 3G, Bluetooth...
- Camera, GPS, Accelerometer, Compass...
The Challenges (cont.)

Constraints

- Low frequency CPU
- Little available RAM
- Low write access on flash disk
- Application-specific lifecycle (energy-afficiency, limited resources)
- Low rate and intermittent networks
- Specific HMI design
  - Small screens: HVGA (320x480)
  - portrait and landscape
  - Small fonts can be unreadable (DPI)
  - Low resolution of tactile sensor (25 pixels)
Downloading and Uploading Apps

Google Play Store (former Android Market)
Standard system to upload/download apps
- No filtering/verification of apps
- Navigation more difficult than AppMarket (search by category, keywords, price)
- Requires a certified terminal (camera, 3G, compass...)
- Authorization management before installation
- 1+ million available apps (57% free-of-charge)

If you want to sell your apps:
- 25$ to register as a developer
- 70% of the app price to the developer, 30% to Google
- Revenue earned from the Google Play is paid to developers via Google Wallet
Other Platforms
There exist other legal application distribution platforms (i.e. authorized by Google). They allow access to non certified devices

- **AppsLib (Archos):** [http://appslib.com](http://appslib.com)
- **AndroLib:** [http://www.androlib.com](http://www.androlib.com)
- **Market Samsung**
- ...
Agenda

Android Overview

Basics

Some Vocabulary
Android Software Architecture

Android Projects

Activities

XML Layout

Manifest

Some Android-specific Classes

References
Important Terms

Activity

- **Building block** for user interfaces
- Equivalent to a window (Windows, Linux) or a dialog box
- An activity may not have any user interface (in case of services)

Content provider

- **Abstraction level** for any data stored in the device
- Android encourages the **provision** of its own data to other programs
- the **content provider** enables this by proposing an **access control** mechanism
Important Terms (cont.)

Intent

- An intent is a kind of system message called event to glue activities
- Emitted by the device to alert applications that a certain event occurred (system event) or by any other app (applicative event).
  - System: insertion of SD card, SMS received
  - Applicative: “App HelloWorld starts”, “The user is arriving in Paris”

Service

- A Service is an application component representing either an application’s desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use
- Does not need any UI
- Examples:
  - Service verifying RSS flux updates
  - Service enabling to listen a given playlist (independently from any activity)
Important Terms (cont.)

**Manifest**
- Starting point for every Android app
- Declares *what the app contains* (activities, services, ...)
- Specifies how components are linked to Android (e.g. what is displayed in the menu?)
- Specifies *permissions* for the app (e.g. webcam, network, location access, ...)

**Widget**
- Portmanteau combining *window* and *gadget*
- Concretely it is a *GUI component* (labels, text fields, buttons, ...)
Important Terms (cont.)

View

- Represents the **basic building block** for user interface components
- Occupies a **rectangular area** on the screen and is responsible for drawing and event handling
- View is the base class for **widgets**

Container (View Group)

- A View Group is a special view that can **contain other views** (called children)
- Used to have a set of widgets to the **desired presentation**
- Most tools to build GUIs provide layout managers that are most frequently organized in containers
Important Terms (cont.)

**XML**
- Extensible Markup Language
- Used to structure data

**XML Layout**
- Used to design GUI easier than using Java
- Used to instantiate widgets, position views and viewgroups
- The XML file is often generated using tools to build graphically GUIs
Uniform Resource Identifier (URI)

- is a string of characters used to identify a name of a web resource (real or not)
- respects the Internet norm RFC 3986

Some URIs:

- **Uniform Resource Locator (URL):** identifies a resource on a network and provides means of acting upon or obtaining a representation of the resource by providing both the primary access mechanism, and the network "location"
  
  Example: [http://www.wikipedia.org/](http://www.wikipedia.org/) identifies a resource (Wikipedia’s homepage) and implies a representation of this resource (HTML page) can be obtained via the HTTP protocol from an network host whose name is [www.wikipedia.org](http://www.wikipedia.org)

- **Uniform Resource Name (URN):** identifies a resource by its name in a specific namespace
  
  Example: [urn:isbn:0-395-36341-1](http://www.wikipedia.org/) identifies a resource by an ISBN number, allows to reference a book, but does not indicate where nor how to obtain a physical copy of it
Important Terms (cont.)

Android Virtual Devices

- AVDs allows to simulate the execution of an Android device on a computer
- These devices are customizable (Android version, CPU model, memory, ...)
  ⇒ Eases the development and the implementation of apps
Agenda

Android Overview

Basics

Android Projects
   Create a project from an existing sample
   Create your own project
   Android Project Directory Structure

Activities

XML Layout

Manifest

Some Android-specific Classes

References
Create a project from an existing sample
Create a project from an existing sample

Browse Samples
Select a sample to import into Android Studio

Sample demonstrating how to set up SensorEventListener for step detectors and step counters.

Tags: sensors

Browse source in GitHub
Create a project from an existing sample

Import Sample
Android Studio

Sample Setup
Provide information about your project

Application name: BatchStepSensor
GitHub URL: https://github.com/googlesamples/android-BatchStepSensor/
Project location: ~/AndroidstudioProjects/BatchStepSensor

Finish
Create a project from an existing sample
Create your own project

Welcome to Android Studio

Recent Projects
- BatchStepSensor
- Aml
- Geofencing

Quick Start
- Start a new Android Studio project
- Open an existing Android Studio project
- Import an Android code sample
- Check out project from Version Control
- Import Non-Android Studio project
- Configure
- Docs and How-Tos
Create your own project

Configure your new project

Application name: My Application
Company Domain: emse.fr
Package name: fr.emse.myapplication
Project location: ~/AndroidStudioProjects/MyApplication
Create your own project

Select the form factors your app will run on
Different platforms require separate SDKs

- **Phone and Tablet**: Minimum SDK - API 18: Android 4.3 (Jelly Bean)

  Lower API levels target more devices, but have fewer features available. By targeting API 18 and later, your app will run on approximately 32.5% of the devices that are active on the Google Play Store. *[Help me choose]*

- **TV**: Minimum SDK - API 21: Android 5.0 (Lollipop)

- **Wear**: Minimum SDK - API 21: Android 5.0 (Lollipop)

- **Glass (Not Installed)**: Minimum SDK
Create your own project

Add an activity to Mobile

- Add No Activity
- Blank Activity
- Blank Activity with Fragment
- Fullscreen Activity
- Google Maps Activity
- Google Play Services Activity
- Login Activity
- Master/Detail Flow
- Navigation Drawer Activity
- Settings Activity
Create your own project

Choose options for your new file

Creates a new blank activity with an action bar.

Activity Name: MainActivity
Layout Name: activity_main
Title: MainActivity
Menu Resource Name: menu_main

The name of the activity class to create

Finish
Android Project Directory Structure
Android Project Files

.idea : Directory for IntelliJ IDEA settings
app : Application module directories and files
build : This directory stores the build output for all project modules
gradle : Contains the gradle-wrapper files
.gitignore : Specifies the untracked files that Git should ignore
build.gradle : Customizable properties for the build system
gradle.properties : Project-wide Gradle settings
gradlew : Gradle startup script for Unix
gradlew.bat : Gradle startup script for Windows
local.properties : Customizable computer-specific properties for the build system, such as the path to the SDK installation
.iml : Module file created by the IntelliJ IDEA to store module information
settings.gradle : Specifies the sub-projects to build
Android Application Modules

build/ : Contains build folders for the specified build variants

libs/ : Contains private libraries

src/ : Contains your stub Activity file, which is stored at src/main/java//ActivityName>.java

androidTest/ : Contains the instrumentation tests

main/java/com.>project<.>app< : Contains Java code source for the app activities

main/gen/ : Contains the Java files generated by Android Studio, such as your R.java file

main/assets/ : This is empty; you can use it to store raw asset files
Android Application Modules (cont.)

**main/res/**: Contains application resources, such as drawable files, layout files, and string values in the following directories

- **anim/**: For XML files that are compiled into animation objects
- **color/**: For XML files that describe colors
- **drawable/**: For bitmap files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states (normal, pressed, or focused)
- **layout/**: XML files that are compiled into screen layouts (or part of a screen)
- **menu/**: For XML files that define application menus
- **raw/**: For arbitrary raw asset files
- **values/**: For XML files that define resources by XML element type
- **xml/**: For miscellaneous XML files that configure application components

**AndroidManifest.xml**: The control file that describes the nature of the application and each of its components
Android Application Modules (cont.)

.gitignore/ : Specifies the untracked files ignored by git

app.iml/ : IntelliJ IDEA module

build.gradle : Customizable properties for the build system
Agenda

Android Overview
Basics
Android Projects
Activities
  Definition
  Activity Lifecycle
XML Layout
Manifest
Some Android-specific Classes
References
What is an Activity?

Definition (Activity)

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

- An Activity is a UI for a use case (à la UML):
  
  Example: Consider a SMS sending app
  
  ▶ One activity to choose the contact
  ▶ One activity to write the message
  ▶ One activity to display the message history

- From an operational point of view:
  
  ▶ An activity must extends the android.app.Activity
  ▶ An activity is independent from the other activities BUT:
    ▶ You must specify a starting activity (the first one to be used)
    ▶ You must chain the activities (an activity activates another one)
  
  ▶ We consider (for simplification) that one activity corresponds to one XML layout
Activity Lifecycle

**onCreate():**

- called when the activity is created
- if this is the first activity of the app, this method is executed when the user call the app
- this method is used to initialize:
  - the XML view
  - if necessary, the temporary files and data
Activity Lifecycle

**onRestart()**: 
- called after the activity has been stopped, just prior to it being started again
- always followed by `onStart()`
- this method is therefore called when the app becomes the foreground app after a long pause
onStart():
- called just before the activity becomes visible to the user
- this method is called after each call to onCreate() or onRestart()
- followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden
- if necessary, it reloads data saved before the previous interruption
**Activity Lifecycle**

- **onResume()**: 
  - called just before the activity starts interacting with the user (foreground) 
  - called after each `onStart()` 
  - at this point the activity is at the top of the activity stack, with user input going to it 
  - always followed by `onPause()` 
  - typically:
    - it manages the connexion to the database 
    - it updates data that may be changed before `onResume()`
onPause():

- Called when:
  - the system is about to start resuming another activity
  - the user asks a `finish()` for this activity
  - the system needs to free some memory

- automatically called before `onStop()`
- typically:
  - commit unsaved changes to persistent data
  - stop animations
  - disconnect the database...

- It should do whatever it does very quickly, because the next activity will not be resumed until it returns
onStop():
- called when the activity is no longer visible to the user (destruction or another activity is foreground)
- followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away
- free resources
**Activity Lifecycle**

**onDestroy()**:  
- called before the activity is destroyed  
- this is the final call that the activity will receive  
- it could be called either because the activity is finishing (someone called `finish()` on it), or because the system is temporarily destroying this instance of the activity to save space  
- `onCreate()` should be called again to launch the activity  
- if necessary free temporary files
Agenda

Android Overview

Basics

Android Projects

Activities

XML Layout
  XML
  Using XML Layouts

Manifest

Some Android-specific Classes

References
Simple XML Example

Example (A sample XML declaration for a library stock)

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<BIBLIOTHEQUE>
  <MAGAZINE>
    <TITRE>Science et Vie</TITRE>
    <DATEPARUTION>01-02-2012</DATEPARUTION>
    <PRIX devise="Euro">4.20</PRIX>
  </MAGAZINE>
  <LIVRE type="education">
    <TITRE>L'art du développement Android</TITRE>
    <AUTEUR>Mark Murphy</AUTEUR>
    <PRIX devise="Euro">32.30</PRIX>
  </LIVRE>
  <LIVRE type="roman">
    <TITRE>Le livre d'Android</TITRE>
    <AUTEUR>Patrick Beuzit</AUTEUR>
    <PRIX devise="Dollar">22.25</PRIX>
  </LIVRE>
</BIBLIOTHEQUE>
```
XML Layout Example

Example (Sample XML Layout)

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@id/actionbar_compat"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView"/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button"/>

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="168dp"
        android:layout_height="0dip"
        android:layout_weight="0.36"
        android:src="@drawable/ic_menu_share"/>

</LinearLayout>
```

**LinearLayout (lines 1 to 5)**

1. `<LinearLayout ... />`: specifies the view group (here widgets will be inline)
2. android:id="...": declaration of the name (id) of the resource
3. android:layout_width="...": the width set to fill the parent (the screen)
4. android:layout_height="...": the height set to fill the parent (the screen)
5. android:orientation="...": the contained views are be vertically lined up
XML Layout Example

Example (Sample XML Layout)

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:id="@id/actionbar_compat"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="168dp"
        android:layout_height="0dip"
        android:layout_weight="0.36"
        android:src="@drawable/ic_menu_share" />

</LinearLayout>
```

**TextView (lines 7 to 11)**

7  `<TextView ... / >`: the view is a label (text)
8  android:id="@+id/text": the id associated to this view is textView1
9  android:layout_width="wrap_content": the width is adapted to the contained text
10  android:layout_height="wrap_content": the height is adapted to the contained text
11  android:text="Hello, I am a TextView": the displayed text
XML Layout Example

Example (Sample XML Layout)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@id/actionbar_compat"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="168dp"
        android:layout_height="0dp"
        android:layout_weight="0.36"
        android:src="@drawable/ic_menu_share" />

</LinearLayout>
```

Button (lines 13 to 17)

13  <Button .../>: the view is a button
14  android:id="@+id/button1": the id of the button is button1 (cf. Java code)

In Java, we can create an instance of class Button to manipulate this view:

```
Button myButton = (Button) findViewById(R.id.button);
```
XML Layout Example

Example (Sample XML Layout)

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@id/actionbar_compat"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="168dp"
        android:layout_height="0dp"
        android:layout_weight="0.36"
        android:src="@android:drawable/ic_menu_share" />

</LinearLayout>
```

**ImageView (lines 19 to 24)**

19  <ImageView ... />: the view is an image
20  android:id="@+id/imageView1": the id of the image view is imageView1
21  android:layout_width="168dp": the width of the image is 168 dot point
22  android:layout_height="wrap_content": the height is adapted to the content
23  android:layout_weight="0.36": see the property android:layout_weight
24  android:src="@android:drawable/ic_menu_share": URI of the image
Integrate Layouts in Your App

Steps

- In your XML editor (e.g. Eclipse with ADT plugin):
  1. Create an XML layout (pay particular attention to the views’ ids)

- Dans le programme Java:
  2. Load the layout
  3. Instantiate the views upon which your want to act
(1) Create an XML layout

Manual Edition

- Create a new .xml file in the directory res/layout
- Edit the layout with an XML editor
(1) Create an XML layout (cont.)

(1.a) Choosing a Layout

- **LinearLayout**: arranges its children in a single column or a single row
- **AbsoluteLayout**: lets you specify exact locations (x/y coordinates) of its children
- **RelativeLayout**: the positions of the children can be described in relation to each other or to the parent
- **TableLayout**: arranges its children into rows and columns

(1.b) Choosing and Positionning the Views
Create an XML layout (cont.)

- **EditText**: an editable text field
- **Toast**: a view containing a quick popup message for the user
- **ImageView**: an arbitrary image, such as an icon
- **CheckBox**: a specific type of two-states button that can be either checked or unchecked
- **Button**: a push-button widget that can be pressed, or clicked, by the user to perform an action
- **RadioButton**: a two-states button that can be either checked or unchecked
- **ImageButton**: a button with an image (instead of text) that can be pressed or clicked by the user
- **DatePicker**: a widget for selecting a date
- **SlidingDrawer**: hides content out of the screen and allows the user to drag a handle to bring the content on screen
(1) Create an XML layout
Creation with Eclipse ADT plugin
(1) Create an XML layout
Creation with Eclipse ADT plugin
(1) Create an XML layout
Creation with Eclipse ADT plugin
(2) Load the Layout

Linking Java Code and Layouts

- When the app is compiled, each XML layout is compiled as a View resource.
- The application code must load the layout in the `onCreate()` method of the activity calling the `setContentView()` method.
- The `setContentView()` method takes as input the reference of the layout (its id), e.g. `R.layout.main_layout`.

Example (A minimal `onCreate()` method loading a layout)

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```
(3) Instantiate the Views

Manipulating views by id

- Create an instance of each object on which you want to add a handler in the `onCreate()` method of your activity
- Use the `findViewById()` to get the instance for your view (using the id defined in the XML layout)
- Manipulate your view (e.g. add an event listener for a button, get the content of an editable text field)

Example (A sample `onCreate()` method adding an event listener to a Button)

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.layout1);
    Button button1 = findViewById(R.id.button1)
        .setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // perform an action on click
            }
        });
}
```
Manifest

- What is the manifest file?
- Contents
  - AndroidManifest.xml
- Example

Some Android-specific Classes

References
What is the manifest file?

Definition (Manifest File)

Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application’s code.

- XML file
- Specify the architecture of the app
- AndroidManifest.xml is at the root of the project
In a glance...

- It names the Java package for the app (the package name \( \equiv \) unique identifier for the app!)
- It describes the components of the app
  - It lists the activities, services, and content providers that the app is composed of
  - It names the classes that implement each of the components and
    publishes their capabilities (e.g. which Intent messages they can handle)
- These let the Android system know what the components are and under what conditions they can be launched
- It determines which processes will host app components
- It declares which permissions the app must have in order to access protected parts of the API (GPS, internet, etc.) and interact with other apps
- It declares the minimum level of the Android API that the app requires
- It lists the libraries that the app must be linked against
- It lists the Instrumentation classes that provide profiling and other information as the app is running (for development issues)
Conventions

- Only two elements are required:
  - `<manifest>`: contains the package, the version... this root englobes the whole manifest
  - `<application>`: describes the app and contains the list of its components

- All values are set through attributes, not as character data within an element

- All attribute names begin with an `android:` prefix (except for some attributes of the root `<manifest>` element)

Resources

- Instead of containing data per se, the manifest refers to resources using the following format: `@[package:]type:name`
  - e.g. `<activity android:icon="@drawable/smallPic"></activity>`

- Resources are stored in the `res` directory of the app
Permissions

- If an application needs access to a feature protected by a permission, it must declare that it requires that permission.
- Specify permission with a `<uses-permission>` element in the manifest.
- There exist several standard permissions:
  - `android.permission.CALL_EMERGENCY_NUMBERS`
  - `android.permission.READ_OWNER_DATA`
  - `android.permission.SET_WALLPAPER`
  - `android.permission.DEVICE_POWER`

- It is also possible to define your own permissions.

Example (A sample manifest file declaring permissions)
<manifest>

  <permission android:name="com.example.project.DEBIT_ACCT" />
  <uses-permission android:name="com.example.project.DEBIT_ACCT" />

  ...

  <application>
    <activity android:name="com.example.project.FreneticActivity"
              android:permission="com.example.project.DEBIT_ACCT"
              ...

    ...

  </activity>
</application>
</manifest>
Intent Filter

- Components advertise their capabilities the kinds of intents they can respond to through intent filters.
- A component may have any number of filters, each one describing a different capability (e.g. a filter to edit an existing document, a filter to initialize a new document, etc.).
- A filter must point to an action.
Example (A sample manifest file)

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eyrolles.android.multimedia"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity
            android:name="CapturePhoto"
            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>
    <uses-sdk android:minSdkVersion="3" />
    <uses-permission android:name="android.permission.CAMERA" />
</manifest>
```
Agenda

Android Overview
Basics
Android Projects
Activities
XML Layout
Manifest

Some Android-specific Classes
References
Deep into the Android-specific classes

Let’s look at the online documentation...

- android.widget.Toast
- android.app.AlarmManager
- android.media.AudioManager
- android.os.AsyncTask
- android.hardware.Camera
- android.hardware.Sensor
Agenda

Android Overview
Basics
Android Projects
Activities
XML Layout
Manifest
Some Android-specific Classes

References
References

Books

The Busy Coder’s Guide to Android Development
Mark L. Murphy

Professional Android 4 Application Development
Reto Meier

Android Cookbook
O’Reilly

Other documents (lectures...)

- Romain Raveaux, Cours Android — Développement et API, Laboratoire L3I, IUT de La Rochelle
- Nazim Benbourahla, several resources at developpez.com
- Philippe Lacomme, Raksmey Phan, Créer des applications Android