# Android Apps maken

**Inleiding : beknopt overzicht van apps in Android**

(lees voluit dit artikel : <http://www.ibm.com/developerworks/library/os-android-devel/> )

Samengevat, Android ...

- is gebaseerd op de Linux kernel 2.6 , ontwikkeld onder invloed van Google, voor  
 de mobiele markt (smartphones). Tablets slechts vanaf versie 3, en versie 4 is zowel voor  
 smartphones als tablets geschikt.

- is gevolgd uit goede ervaring met mini-linux server in bvb. routers (html-setup pagina)

- gaat een stap verder :   
 - heeft een eigen GUI (en heeft dus een scherm nodig ;-)

- touch control (niet essentieel, je kan Android ook met een muis besturen !)

[zie demo VMWare + android-x86, BlueStacks= “app-player” voor Windows]

- is een “gelaagd” operating systeem, kent o.a. **Windows, Views en Widgets**

**-** heeft GUI elementen (de Views), bvb. **edit boxes, buttons, lists, drop-down lists**, en nog  
 veel meer.

- heeft een ingebouwde browser (gebaseerd op de Safari browser van iOs !), al wordt nu ook  
 standaard de Chrome browser meegeleverd (Firefox, Opera, … kan je zelf installeren)

- ondersteunt meerdere draadloze protocollen : WiFi, Bluetooth, NFC, 3G, 4G …

- kan via GPS “location based services” aanbieden

- heeft nog andere “input” : magnetische sensoren (🡪kompas), accelerometer, camera, …

- heeft 2D en 3D graphics via OpenGL library

- biedt ondersteuning voor de open source SQLite database

Opbouw van de diverse lagen in Android :



(bron : <http://en.wikipedia.org/wiki/File:Android-System-Architecture.svg> )

Android kent al een hele geschiedenis , en vele versies … de versie krijgt naast een (saai) nummer, ook een alfabetisch oplopende naam, die naar een (Amerikaans) dessert of snoep verwijst (intussen is versie 4.4 KitKat al aangekondigd) :



(bron: <http://www.knowahead.in/dyk/androids-versions-are-named-in-alphabetical-order/> )

En hier vind je meer details van elke versie : <http://www.compiletimeerror.com/2012/12/android-os-versions_25.html#.UojD2SfDux4>

Applicaties (apps) voor Android worden geschreven in Java , en draaien op de **Dalvik Virtual Machine,** deze is helaas niet compatibel met de desktop versie (JVM).

(één van de redenen is de ARM-processor die in mobiele devices gebruikt wordt)

Er zijn 4 **soorten Apps** :

* **Activity** : een app met een zichtbare User Interface (UI)
* **Service** : een proces dat (onzichtbaar) blijft draaien op de achtergrond (vb: checken op updates, netwerk monitoren, data ontvangen)
* **Content provider : vergelijk het met een database server, voor apps waar veel data in nodig is, of waar meerdere apps dezelfde database moeten gebruiken**
* **Broadcast receiver : een tussenstap die een app kan starten als bvb. een mail ontvangen wordt, kan bvb een bericht plaatsen in het meldingenvak   
  (*notification area, kan je openen tot de notification drawer*)**

**Welke Ontwikkeltools heb je nodig ?**

Overzicht :

- de “lange weg” : **Eclipse met de officiële Android Developers Tools (ADT) plugin + de Android Virtual Device Manager**, we zullen verder deze tools gebruiken

(opm : binnenkort komt er een geheel nieuwe IDE uit : *Android Studio*)

andere kan je zelf eens proberen (en laat mij weten of het de moeite is !):

- **Xamarin** : gebruikte taal is C# : <http://xamarin.com/>

- **App Inventor** van MIT : <http://appinventor.mit.edu/>

- op Android tablet zelf via de app **AIDE** (zoeken via PlayStore)

Voor de **Eclipse** methode heb je nodig :

* **Java SDK** (deze heb je normaal reeds staan, volgens sommige bronnen zou je wel maar ten hoogste Java SDK 1.6 mogen gebruiken, en niet 1.7 ???)
* **Android SDK :** deze bevat o.a. *android.jar* met alle nodige klassen om een app te bouwen,   
  *docs* (lokaal maar ook online), *samples*, *tools* (met oa **adb** = android debug bridge), usb\_driver (om de app vanuit de pc op je device te installeren & live te debuggen)
* **Eclipse :** dit is de basis-IDE voor Java, C++, enz …
* **Android Development Tools (ADT):** dit is een plugin voor Eclipse om de Android SDK te kunnen gebruiken (en te configureren via de *Android SDK Manager*).

Je kan alles heel eenvoudig installeren door slechts 1 “**bundle**” te downloaden, dan heb je al het nodige in 1 download. Alle instructies vind je hier:

<http://developer.android.com/sdk/index.html>

After agreeing to the license and choosing the correct architecture (32-bit or 64-bit) your download should start. Create a directory called “android” on your C-drive (or D-drive) and copy the zip-file there. Extract it and traverse to the eclipse directory. You can now run the eclipse executable.

Next open the Android SDK manager which can be found under the Window menu. Install the additional packages as shown in Figure 1.

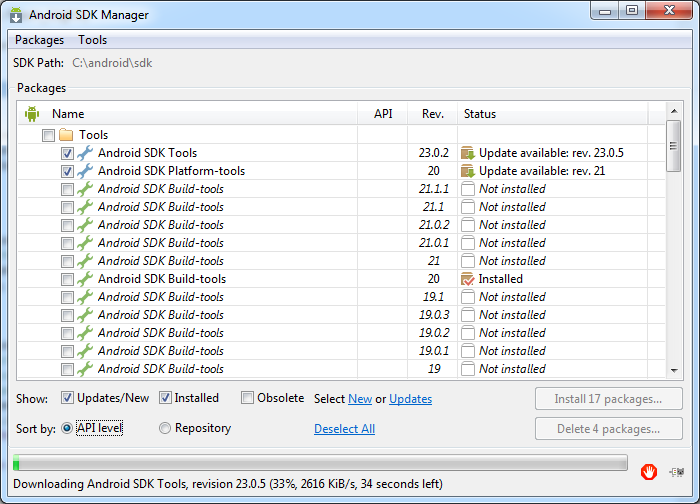


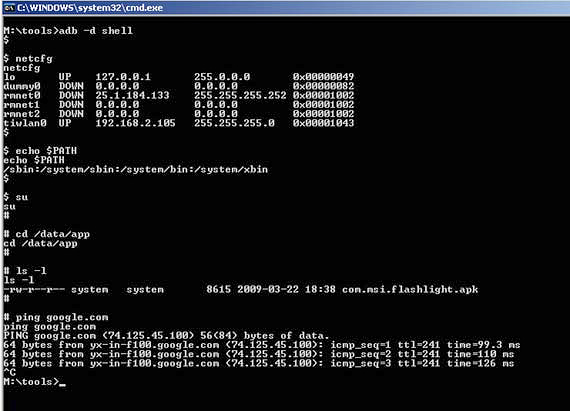
Figure Android SDK manager

Met de meegeleverde emulator (je zult wel een device moeten kiezen), kan je alle apps ook testen zonder over een echt Android toestel te beschikken, je krijgt dan bvb. dit te zien :

(bron : <http://www.ibm.com/developerworks/library/os-android-devel/> )



Met de ADB (Android Debug Bridge) kan je allerlei command-line opdrachten geven aan een echte device, verbonden via een USB-kabel:



(bron : <http://www.ibm.com/developerworks/library/os-android-devel/> )

(het vervolg van de tekst is voornamelijk gebaseerd op dit boek : **Beginning Android 4 Development – Wei-Meng Lee, WROX,** ISBN 978-1-118-19954-1 ,   
maar ook als gratis e-book te vinden op <http://it-ebooks.info/book/1313/> )

**Wrox H1: Software Installeren - Een eerste App maken en bekijken**

Om je apps in een virtuele device te testen moet je via de AVD Manager een nieuwe AVD maken, uitleg op blz. 18.

Wil je liever “live” op een echte device testen, zie verderop voor wat extra uitleg.

Zie demo in de les, we maken een variatie op de klassieke HelloWorld app … we zullen de onze de originele naam HelloVives geven ;-)

(dit is ook de naam van het project, waar we verderop met <naam> naar verwijzen)

Maak deze eerste app en test hem uit.

Beschrijving in detail zie boek p20-28

**Emulator te groot voor je scherm ?**

Ga naar Run > Run Configurations > tab Target, onderaan command line optie bijvoegen

-scale 0.50 (bvb voor 50% van originele grootte)

**Bekijk de opbouw van de diverse files (p 29)**

In de Package Explorer zie je o.a. volgende mappen en files :

* src: de <naam> .java broncode
* gen: generated code, R.java , deze file bevat referenties naar alle resources in het   
   project, wordt door de compiler gegenereerd en mag je dus NIET wijzigen
* Android 4.x: bevat de android.jar, dus alle classfiles van het project.
* assets: bevat alle assets , dit zijn “niet-java” bestanden (html, tekst, databases, enz)
* bin: bevat de binaries die gegenereerd worden, o.a. <naam>.**apk** (Android Package)
* res: alle resources, verdeeld in meerdere submappen, om bvb. string literals te bewaren,  
   icoontjes in meerdere resoluties (zie mappen ldpi/mdpi/hdpi), … 🡪zie H3
* *AndroidManifest.xml* : bevat o.a. de nodige info voor een correcte installatie van de app , op welke events de app kan reageren, de nodige “permissions” die de app nodig heeft, … 🡪zie H2

**String values (p 30)**

Gebruik in je code liever geen “hardcoded strings” (of string literals), maar wel verwijzingen naar strings die je in een apart bestand plaatst. Dit heeft het voordeel dat je een app dan eenvoudig naar een andere taal kan vertalen.

Alle gebruikte strings vind je dan onder res\values\strings.xml

Een referentie naar een string ziet er bvb zo uit:

*android:tekst=”@string/message”*

in strings.xml vind je dan :

*<string name=”message”>Dit is het eigenlijke bericht</string>*

**Manifest bestand** (bekijk het bestand in source view) **(p 31)**

Het manifest bestand bevat een aantal zaken :

* De naam van de package
* Het versie nummer (van onze applicatie)
* De minimale SDK versie waarop de app zal draaien
* Het icoon van de app (uit de map res\drawable)
* De naam van de app (*app\_name* uit res\values\strings.xml)
* Het entry-point van de app, vervat in het element <intent-filter>, dit zegt hoe de app kan gestart worden en waar het beginpunt van de app is

**Het hoofdprogramma <naam>.java (p 33)**

Dit bevat voorlopig 1 methode onCreate()

Daarin wordt 1 andere belangrijke methode opgeroepen : setContentView(**R.layout.main**), dat de app op het scherm brengt.

**R.layout.main** is hierin een referentie naar het bestand **main.xml**, waarin de layout van de UI beschreven staat.

Opm : het is de gewoonte om in Android de UI (.xml) van de code (.java) gescheiden te houden, maar je kan in principe ook alle Views (knoppen, tekstvakken, enz) genereren vanuit het codebestand (zie p 146).

**Wrox H2:** **Activities – Intents - Fragments**

Eerst : **Life cycle van een activity (p 36)**



Basis activity bevat enkel de onCreate() methode, maar in de life cycle zijn nog volgende methodes van belang (zie figuur):

* onCreate()
* onStart()
* onResume()
* onPause()
* onStop()
* onDestroy()
* onRestart()

**Voorbeeld Activity101 (p 38)** : toont hoe je kan loggen in welke methoden je activity terecht komt, gebruikt de klasse **Log**

**Voorbeeld Dialog (p 42)**:een dialoogvenster met enkele keuzes, in het thema Theme.Dialog: drie varianten: 1x gewone Dialog, 2x ProgressDialog

Hierbij komen twee belangrijke klassen ter sprake :

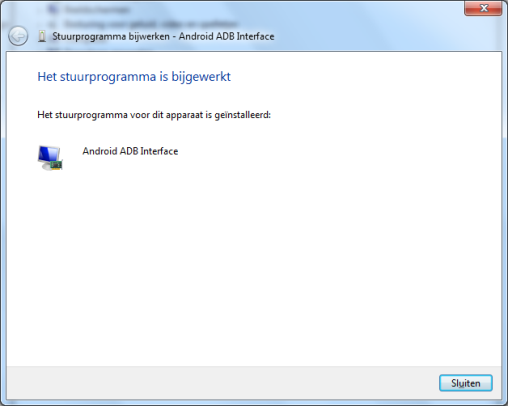
**Context** : instanties hiervan verwijzen naar de eigen applicatie (this)

**Toast** klasse : tijdsgebonden, kleine popups voor korte mededelingen of bevestiging

(vb : “message sent”). Nemen niet meer plaats in dan de tekst die ze bevatten, de lopende activity blijft zichtbaar. Verdwijnen vanzelf na een korte tijd.

**Live debuggen op je Android Device**

Een beperkt aantal toestellen zijn “Google Developer Phones”, deze werken met de Google USB driver die je vanuit Eclipse kan installeren (via de SDK Manager).

Voor de meeste andere devices (waaronder ook de Nexus 7 van Google zelf !!!)moet je een “OEM” USB driver zoeken, meer uitleg hier:

<http://developer.android.com/tools/extras/oem-usb.html>

Als de driver goed geïnstalleerd is , lees je hier verder hoe je op een echte device kan debuggen:

<http://developer.android.com/tools/device.html>

Volgens bovenstaande site moet je normaal in Androidmanifest.xml zetten :

<application

android:debuggable="true"

Dit hoeft echter NIET, Eclipse laat standaard debugging toe.

Je hoeft dan ook achteraf deze lijn niet te verwijderen om ,de app te publiceren.

**Op je device** moet je wel enkele instellingen veranderen, als dit niet reeds gebeurd is:

* Beveiliging > Onbekende bronnen: aanvinken = toelaten dat je ook apps van “externe bronnen” kan installeren.
* { } Opties voor ontwikkelaars > USB-foutopsporing: aanvinken  
  [vind je geen developer options ? 🡪 ga naar *Over tablet/telefoon* > *Build nummer* en tap er een aantal keer op tot een mededeling verschijnt]

(\* bij de Nexus 7 nog wijzigen via notification area : verbonden als *media device* naar > verbonden als *camera*, als deze notification verdwenen is, terug halen via Instellingen > Apparaat > Opslagruimte > ⁞ > USB-computerverbinding)

Laat Eclipse debuggen zoals anders, je krijgt een scherm waarin je de device kan selecteren: de emulator of de échte Android device.

(\* dit werkte niet bij mij : na alle avd’s te wissen werkte het wel)

**INTENTS (p 53)**

Is een manier om in een app naar een andere activity binnen (of buiten) de app over te schakelen.

Typische code in hoofdprogramma:

startActivity( new **Intent** (“be.vives.Tweede”));

//( Tweede.java bevat dan de code voor die tweede activiteit)

In het manifest moet hiervoor een intent filter bijgeplaatst worden.

**INTENT Collision (p 58)**

Stel een derde activity heeft als filter dezelfde naam als de tweede, dan komt een selectiemogelijk welke je wil gebruiken (antwoord “altijd” of “één keer”).

(heb je “altijd“ geantwoord, dan wordt dit onthouden, maar je kan die keuze weer wissen via Instellingen>Apps>App kiezen>Standaard starten>wissen)

Resultaten **terugkrijgen** **uit** een andere activity (vb ingetikte naam) **(p 59)**:

startActivityForResult( .. )

Gegevens **zenden** **naar** een activity **(p 63)**:

Gewoon een String doorgeven kan direct in het Intent object :   
in sub : i.setData(Uri.parse (“Gelukt!”)

In main : i.getData().toString()

Er zijn nog twee andere systemen:

* direct in een Intent object ( putExtra() ),
* eerst in een Bundle object ( vb. putString() , putInt(), enz …) , dat dan in het Intent object gevoegd wordt ( putExtras() ). Dit zijn telkens **name/value pairs** (koppels gegevens).

In de sub-activity kan je aan die data met opdrachten als getIntent().getStringExtra, getIntent().getIntExtra() , enz …

**FRAGMENTS (p 69)**

Met komst van de tabletskwamen er ook grotere schermen, waar je meerdere “panelen” in 1 activity kan weergeven = **fragments**.

* Maak voor elk fragment een xml-bestand en een java klasse aan (extends Fragment)

**Dynamisch Fragments toevoegen** **(p 73)**: FragmentManager + FragmentTransition

Stand van device te bepalen via de WindowManager > Display object opvragen en daarvan getHight() + getWidth() gebruiken (deprecated : je zou nu getRotation() of getSize() moeten gebruiken … nog te testen).

**Life cycle** van een fragment **(p 76)**

**Interactie tussen Fragments (p 80)**

Geef een label mee aan de Views van een Fragment, die je wenst te gebruiken vanuit een ander Fragment.

Vb de TextView in Fragment1 krijgt label lblFragment1

Maak in de xml van F2 een Button, die verwijst naar onClick()

In de bijhorende Java code die methode invullen.

In main.xml de twee fragmenten terug activeren.

**Ingebouwde applications oproepen via Intents (p 85)**

Voorbeeld **Intents** :

Drie knoppen roepen op : browser, call manager, maps

**Gebruik van Intent Filters (p 91)**  zodat jouw activity ook door andere activities kan opgeroepen worden.

Demo met een vierde knop om een browser op te roepen (MyBrowserActivity)

**Toevoegen van Categories (p 96)**

Als de juiste categorie toegevoegd werd in AndroidManifest.xml:

<category android:name="net.learn2develop.Apps" />

en in je programma ook aan een Intent werd toegevoegd:

Intent i = new Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse(“http://www.vives.be”));

i.addCategory(“net.learn2develop.Apps”);

dan kan je de activity starten via deze categorie met :

startActivity(Intent.createChooser( **i** , “Open URL using...”));

Een activity die deze categorie niet heeft, zal niet starten.

**Notifications (p 98)**

In plaats van de korte Toast mededelingen (pop-ups) kan je een meer permanente mededeling doen via de *notification drawer.*

Dit kan via een object van de klasse *NotificationManager,* Zie voorbeeld **Notifications** (p98)

Er zijn twee verschillende mededelingen: een tekst voor de “ticker tape” (komt even op de stausbalk), en de tekst die in de notification drawer blijft staan.

Je kan ook de trilfunctie van het toestel gebruiken.

SAMENVATTING van dit hoofdstuk : **p 104**

**Wrox H3:** **Meer over UI-ontwerp, Layouts, Views en Viewgroups, Eventhandlers (p 105)**

**Views** *(android.view.View)* **(p 106, p 109)**

De UI (user interface) van een activity bevat **widgets** : labels, knoppen, vakken, enz …

Dit zijn allemaal “Views”.

Verwijzen we even naar Java, dan zijn dit enkele equivalenten **:**

|  |  |
| --- | --- |
| **JComponent** | **View** |
| JButton | Button |
| JLabel | TextView |
| JTextField | EditText |

enz … (todo : eens aanvullen ?)

De UI wordt standaard in een xml bestand opgebouwd (in de map /res/layout), maar kan ook volledig vanuit Java code gemaakt worden (oa. wanneer je dynamisch views wil toevoegen).

Afmetingen van views worden uitgedrukt in de eenheid **dp (**of **dip**, dit is een *density independent pixel).*Op iedere device wordt dan geprobeerd een gepaste grootte voor je views te nemen.

De resolutie van een device wordt afgerond naar de dichtstbijzijnde standaard :

|  |  |  |
| --- | --- | --- |
| resolutie | Echte dpi | Nodige gif in res\\*dpi |
| ldpi | 120 | 36 x 36 |
| mpdpi | 160 | **48 x 48** |
| hdpi | 240 | 72 x 72 |
| xhdpi | 320 | 96 x 96 |
| xxhdpi | 480 | (144 x 144)\* |

De Nexus 7 (resolutie 1280x800 voor een 7” diagonaal) valt bvb in de categorie 240 dpi.  
48 x 48 pixels is de standaard resolutie voor een “app launcher” icoontje.

\* is nog niet officieel vastgelegd

Je kan ook de eenheid **px** geb ruiken, dit zijn absolute pixels, maar dan krijgen je views een verschillende ware grootte op verschillende devices. Niet aan te raden, tenzij je bvb. een meetlat in cm op het scherm wil zetten, zie bvb de app “Swiss Army Knife”.

Je kan wel berekenen met hoeveel pixels een view zal weergegeven worden op een bepaalde device, met deze formule :

**px = dp \* (dpi/160)**

De afmeting van een view wordt in de xml aangeven met bvb :

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

In dit geval neemt de view in de breedte alle plaats in die beschikbaar is, in de hoogte enkel de nodige ruimte om de inhoud van de view (vb. de tekst) weer te geven.

Enkele andere mogelijkheden, vb :

android:layout\_width=*"160dp"*

Op een Nexus 7 zal deze view dus niet met 160 maar met 240 pixels weergegeven worden

( 160\* (240/160)).

Een andere eenheid die je veel nodig hebt is de **sp** (scale independent pixel).

Hiermee wordt meestal de FONT ingesteld, bvb :  
 android:textSize="18sp"

Als je afbeeldingen gebruikt in een app (via een *ImageView*), dan ka je best de figuur in de verschillende resoluties aanmaken, in de mappen **\res\drawable-ldpi**, enz …

Je moet minstens in de map mdpi een figuur zetten, de andere zijn optioneel.

**ViewGroups** *(android.view.ViewGroup)* **(p 106)**

Dit zijn de “layout managers” van Android. Het zijn de *parents* voor één of meerdere *children*. (zie ook <http://developer.android.com/reference/android/view/ViewGroup.html> )

De meest gebruikte zijn LinearLayout en RelativeLayout. Andere zijn nog : AbsoluteLayout, TableLayout, FrameLayout.

Van die laatste, FrameLayout, stamt ook nog de ScrollView af, waarmee je meer views kan groepren dan er op een scherm kunnen, niet te verwarren met een ListView , die 1 component is die meerdere items kan weergeven in een lijst.

**LinearLayout (p 107)**

Rangschikt de elementen in 1 kolom of 1 rij, volgens het attribuut “orientation”

android:orientation=”vertical” … of "horizontal"

andere attributen, zie tabel p 107

**Gravity en Weight (p 113)**

Bij LinearLayout kan je de onderlinge grootte ook bepalen met deze twee attributen.

Vb. drie knoppen, met resp :

android:layout\_gravity="left" .. "center" .. "right"

android:layout\_weight= "1" .. "2" .. "3"

Horizontaal staan de views dus links , in het midden en rechts.

Verticaal wordt de ruimte gedeeld door 6 (= 1+2+3), en vervolgens verdeeld over 1/6, 2/6 en 3/6, zodat heel de ViewGroup in de hoogte gevuld is (zie figuur 3-9).

**AbsoluteLayout (p 115)**

Verouderd, niet meer gebruiken !

**TableLayout (p 116)**

Views groeperen in rijen en kolommen.

Iedere rij begint met de tag <TableRow …, daarin plaats je dan de views van die rij.

De breedte van de elementen in de eerste rij bepaalt de kolombreedte, die hoef je dus niet telkens te herhalen, tenzij je een andere indeling wil.

**RelativeLayout (p 117)**

Hiermee geef je de plaatsing van de views t.o.v. elkaar aan, vb :

<Button

...

android:layout\_below="@+id/txtComments"

android:layout\_alignRight="@+id/txtComments" />

De knop komt dus rechts onder het tekstveld txtComments geplaatst (zie figuur 3-15)

**FrameLayout (p 118)**

Hierin komen alle views steeds dezelfde positie, nl. linksboven

Voeg je meerdere views toe aan de FrameLayout, dan komen ze boven elkaar te liggen, met de laatste view bovenaan.

Toepassing is bvb een knop op een figuur plaatsen, zodat het lijkt alsof je op de figuur kan klikken (zie figuur 3-16).

**ScrollView (p 121)**

Dit is een uitbreiding op de FrameLayout, voor het geval je méér views hebt dan er op het scherm kunnen. De ScrollView kan slecht 1 View of 1 ViewGroup bevatten, meestal plaatst men een LinearLayout in een ScrollView.

Opm : gebruik deze view niet samen met een ListView !

Zie voorbeeld in het boek p121 : een LinearLayout, gevuld met drie Buttons, een EditText van 600 dp hoog, en dan nog twee knoppen. Normaal komt enkel het tekstvak op het scherm, daarom zijn enkele bijkomende “trucjes” nodig :

* tekstvak niet automatisch de focus geven 🡪 voeg toe in LinearLayout :  
   android:focusable="true"  
   android:focusableInTouchMode="true"
* tekstvak wel de focus geven, maar niet automatisch het on-screen keyboard tonen 🡪 in AndroidMAnifest.xml dit toevoegen :  
   android:windowSoftInputMode="stateHidden"

**\*\*\* TODO : GridLayout (niet in boek)**

**Display orientation (p 123)**

Om een app automatisch te laten omschakelen tussen *portrait* en *landscape* is volgende code nodig.

Bij het wijzigen van de stand van een device wordt telkens automatisch *onCreate()* opnieuw opgeroepen, om de activity te hertekenen (de activity wordt eigenlijk vernietigd en opnieuw opgestart.

Merk op dat zomaar hertekenen niet steeds het gewenste resultaat geeft, zie figuur 3-20.

De views worden hertekend alsof de device in *portrait* staat, waardoor rechts veel verloren ruimte ontstaat.

Er zijn twee technieken om dit netjes op te lossen: **anchoring**  en  **resizing & repositioning.**

**Anchoring Views (p 125)**

Hierbij “hang” je de views aan de vier hoeken van het scherm.

Dit wordt gedaan in een RelativeLayout, zie attributen, bvb.

Bovenaan links :

android:layout\_alignParentLeft="true"

android:layout\_alignParentTop="true"

In het midden van het scherm:

android:layout\_centerVertical="true"

android:layout\_centerHorizontal="true"

**Resizing & Repositioning (p 127)**

Hierbij moet je twee keer de layout maken, en in een aparte map plaatsen.

Voor portrait staat een main.xml in \res\layout (omdat portrait de originele stand is voor een telefoon). De layout voor landscape heet ook main.xml, maar plaats je in \res\layout-land.

In het voorbeeld in het boek komen er in landscape nog twee knoppen bij, bovenaan en onderaan in het midden (zie figuur 3-25).

Het laden van het juiste xml-bestand gebeurt automatisch !

**Gedrag van een activity als de oriëntatie wijzigt (p 130)**

Zoals eerder gebruikt kan je met *Log.d()* opdrachten volgen in welke methoden je activity terecht komt, zie p 132. Hieruit blijkt dat de activity vernietigd wordt als je de device kantelt … dus zal je moeten zorgen dat je de variabelen die je wenst te behouden voordien ergens opbergt !

Dit doe je best in de methode *onPause()*.

Er is ook nog een voorwaarde : views waarvan je de inhoud wil bewaren moeten een ID gekregen hebben (vb de tekst die je reeds in een EditText zou ingetikt hebben).

**Status van een activity bewaren (p 133)**

**Dit kan op drie manieren :**

In de *onPause()* methode, deze wordt altijd opgeroepen als een activity naar de achtergond gaat (destroyed of niet). Je moet zelf een database aanmaken , of je data in een intern of extern bestand opslaan.

Iets handiger is in de methode *onSaveInstanceState().* Deze wordt echter niet opgeroepen als de gebruiker zelf de activity beëindigde met de “back”-knop. Voordeel is hier wel dat deze methode standaard een *Bundle* argument heeft om data in op te bergen. Vb, als dit argument de naam *outState* heeft :

outState.putString(“ID”, “1234567890”);

In de *onRestoreInstanceState()* methode haal je dit weer op met:

String ID = savedInstanceState.getString(“ID”);

Dit is reeds een handig system, maar beperkt zich tot name/value koppels die je in een Bundle kan stoppen.

Daarom kan je ook nog de methode *onRetainNonConfigrationInstance()* gebruiken.

Hierin kan je eender welk Object type in een return opdracht zetten, vb

return(“Een String om te bewaren”);

// hier een String literal, maar kan eender welk object zijn

Je krijgt dit object terug als je in *onCreate()*zet :

String str = (String) getLastNonConfigurationInstance();

// of een ander object type, wel telkens typecasten

**De oriëntatie van het scherm bepalen (p 135)**

Als de breedte van het scherm > hoogte, sta je in landscape, anders portrait.

Breedte en hoogte kan je opvragen via de WindowManager + een daarvan afgeleid Display object.

**Een oriëntatie forceren (p135)**

Vb : de activity mag enkel in landscape uitgevoerd worden:

plaats in onCreate() :

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE);

Je kan ook volgend attribuut aan de activity toevoegen in AndroidMAnifest.xml :

android:screenOrientation=”landscape”

In beide gevallen zal de activity niet reageren op een wijziging in oriëntatie, en dus ook niet “destroyed” worden.

**De ACTION BAR tonen of verbergen (p 136)**

De Action Bar staat bovenaan de applicatie , toont het icoontje, de titel en een of meer “action items”, zoals de prullenbak, file explorer, enz …

De actionbar gebruiken : zie p 137, voorbeeld **MyActionBar**

Action bar verbergen : in xml - in Java code

**Action Items toevoegen aan een Action Bar (p 139)**

Er is vrij veel code toe te voegen om eerst een menu te maken met onCreateOptionsMenu (), waarin je CreateMenu() oproept waarin je dan de diverse items opsomt.

De methode onOptionsItemSelected() handelt dan de acties af via een switch statement (zie p 143).

Action Items toevoegen mét bijhorende tekst : zie p 144

Het Application icoontje kan ook aangeklikt worden, dit is dan

case android.R.id.home: enz … (zie p 144-145)

**Views aanmaken vanuit Java code (p 146)**

In dit geval mag je in onCreate() de lijn

setContentView(R.layout.main);

NIET gebruiken, de main.xml wordt dus niet geladen.

In plaats daarvan moet je voor elke view de nodige Java code schrijven, zie voorbeeld p 147

**EVENTS afhandelen (p148)**

Key events : zie p 150

Buttons : zie p 153

SAMENVATTING van dit hoofdstuk : **p 157**

**Wrox H4: Gevorderde Views gebruiken in de User Interface (p 159)**

Naast de basis views Button, TextView en EditText uit hoofdstuk 2, komen nu de meer gespecialiseerde views aan bod, met name :

**Picker views** : TimePicker, DatePicker, …

**List views** : ListView, SpinnerView, …

**Speciale Fragments** …

**Basic Views**

Reeds gekend uit H2 : TextView , EditText , Button

Nieuw : ImageButton = een Button met een “prentje”  
 CheckBox = soort Button die “aangevinkt” wordt (of niet)  
 ToggleButton = soort Button die aan/uit toestand toont met een lampje  
 RadioButton en RadioGroup = de gekende “rondjes”, waarbij er slechts één   
 tegelijk kan actief zijn

Details : zie voorbeeld **BasicViews1.**

Merk op : een EditText met attribuut android:layout\_height=”wrap\_content” past zich automatisch aan als je er meer tekst in typt dan er op één regel kan staan (zie p 164).

Een checkbox hoeft niet steeds een saai “vinkje” te zijn, je kan er bvb volgend attribuut aan geven : style=”?android:attr/starStyle”

Dan krijg je bvb een ster als checkbox (zie p 164).

De vorm voor de stijl is : ?[package:][type:]name

Alle views hebben een @+id/ gekregen, waardoor je in de java code hun referentie kan opvragen met View.findViewById().

**Oudere versies ? (p 166)**

Beschrijving wat er gebeurt als je een app met bvb. target = Android 4.0 (versie 14) laat draaien op een oudere versie.

**Events afhandelen (p 168)**

De events voor al deze views in java code afhandelen.

Zie wijzigingen in het voorbeeld **BasicViews1.**

Voor CheckBox en ToggleButton gebruik je steeds de isChecked() methode (result = boolean). Voor een RadioGroup met RadioButton views kan je ook de isChecked() toepassen, ofwel het argument checkedId van de methode onCheckedChanged() gebruiken.

En tot slot : deze extra methode is handiger om boodschappen weer te geven (p 169) :

private void DisplayToast(String msg)

{

Toast.makeText(getBaseContext(), msg,

Toast.LENGTH\_SHORT).show();

}

Voor alle events wordt het systeem met findViewById() gebruikt.

Er is echter een veel eenvoudiger methode :

Zie voorbeeld met de Save button (p 171)

In de xml geef je aan de knop het attribuut android:onClick=”btnSaved\_clicked”, en in java maak je de methode public void btnSaved\_clicked (View view) { … }

[\* opm : geen idee waarom in het boek dan eerst de “ingewikkelde” methode aangeleerd wordt … ]

**ProgressBar (p 172)**

Zie voorbeeld **BasicViews2.**

Hierbij wordt een **Thread** gebruikt ! In dit geval weliswaar in een *naamloze instantie* van een *anonieme klasse* (zie Laan H11 vorig jaar …).

Normaal krijg je een rondje als “progress indicator”, maar met een stijl attribuut

style="@android:style/Widget.ProgressBar.Horizontal"

kan je er ook een echte “progress bar” van maken.

Meer varianten vind je op p 177.

**AutoCompleteTextView (p 177)**

Een handige subklasse van een EditText (maar heet toch AutoComplete**TextView !**).

Zie voorbeeld **BasicViews3.**

Een array van Strings bevat een lijst met suggesties, als je begint te typen verschijnen een aantal passende namen. Je moet echter wel reeds een minimum aantal karakters getypt hebben (hier = 3, zie opdracht textView.setThreshold(3);)

Klik een van de suggesties aan en die wordt overgenomen in de EditText.

Koppeling met de array gaat via de opdracht textView.setAdapter(adapter);

**Picker Views (p179)**

Datum en tijd instellen kan via een TimePicker en DatePicker.

Zie voorbeeld **BasicViews4.**

**TimePicker**

Voor gebruik in Europa zal je best deze opdracht gebruiken :

timePicker.setIs24HourView(true); (geen AM/PM zoals in de USA)

De gekozen tijd krijg bekom je via de methoden getCurrentHour() en getCurrentMinute(). Een tweede versie van het programma toont hoe je de tijd via een aparte dialoog instelt (p 181).

**DatePicker (p 184)**

Verloopt op gelijkaardige wijze, eerst zónder, dan mét dialoog.

**ListView (p 191)**

Weergave van een lange lijst elementen. Hierbij zijn er twee mogelijkheden : ListView (p 191) en SpinnerView (p 199).

**ListView (p 191)**

Zie voorbeeld **BasicViews5.**

Let bij de java klasse op de … extends ListActivity

In de **eerste versie** maak je de lijst in een array van Strings.

Er is geen xml layout nodig, dus ook geen setContentView() oproep in onCreate(), wel een oproep van setListAdapter(). De methode onListItemClick() reageert op een klik op een van de elementen in de lijst, het element verschijnt op een Toast.

In een **tweede versie** wordt de ListView verder verfijnd (p193) : meerdere elementen kunnen selecteren (aangeduid met een vinkje) met setChoiceMode().  
Met setTextFilterEnabled(true) kan je enkele karakters intikken en de lijst toont enkel de elementen die hier mogelijks mee overeenkomen.

[\* keyboard tip : zie onderaan]

In een **derde versie** wordt getoond hoe je de lijst met elementen beter in **strings.xml** opbergt dan in een array in de sourcecode (p195), zie de tags <string-array> en <item>. De array in de source wordt dan gevuld uit de elementen uit het xml-bestand via een getResources() opdracht.   
Opvragen welke item(s) geselecteerd staan doe je door alle elementen in de lijst te overlopen en telkens met de opdracht isItemChecked(i) de toestand op te vragen.

De naam op die positie kan je opvragen met getItemAtPosition(i).

**Spinner (p 199)**

Dit is een manier om een lange lijst op een kleinere plaats weer te geven ,1 element tegelijk. Dit is dus gewoon een “drop-down” list.

Zie voorbeeld **BasicViews6**.

Merk op dat dit terug een …extends Activity klasse is, en bijgevolg de Spinner view in xml gedefiniëerd is, en met een setContentView() oproep in onCreate() op het scherm geplaatst wordt.

Voorzie ook de methode onNothingSelected() voor het geval de gebruiker de “Back” toets drukt zonder iets te selecteren.

Een andere weergave van de lijst kan met radiobuttons (p 201).

**Gespecialiseerde Fragments (p 202)**

Zie hoofdstuk 2 : fragments zijn een soort mini-activities, waardoor je een app op devices met verschillende schermgrootte toch goed kan weergeven. Elk fragment heeft een eigen life-cycle.

Van de klasse Fragment zijn verschillende speciale subklassen, waarvan je jouw klasse kan afleiden, bedoeld voor enkele speciale toepassingen.

Hier worden drie voorbeelden bekeken : ListFragment, DialogFragment en PreferenceFragment.

**ListFragment (p202)**

Een fragment met een ListView. Zie voorbeeld **ListFragmentExample**.

Hier wordt twee keer hetzelfde fragment naast elkaar getoond, maar met verschillende hoogte (resp. 200 en 300 dp). Merk op hoe er links minder namen staan dan rechts, maar in beide fragmenten kan je door de lijst scrollen.

Let op de class Fragment1 extends ListFragment.

**DialogFragment (p 207)**

Dit is een fragment dat “boven” een activity kan zweven (een pop-up dialoog dus).

Hiervoor moet je een subklasse maken van DialogFragment.

Zie voorbeeld **DialogFragmentExample**.

Het hoofdprogramma DialogFragmentExampleActivity.java laat een dialoog verschijnen maar doet verder niks. De dialoog wordt afgehandeld in Fragment1.java.

**PreferenceFragment (p 210)**

Dit is ook een soort pop-up dialoog, maar met gelegenheid om iets in te vullen.

Zie **PreferenceFragmentExample**.

Hierbij komt een nieuw bestand ter sprake: preferences.xml in een nieuwe, te maken map **\res\xml** (inhoud van dit bestand p 211).

Dit vormt een lijstje “preferences” die de gebruiker kan editeren : ofwel een gewone CheckBox, een tekst die via een pop-up dialoog ingetikt wordt, of nog in een tweede scherm, ook via een pop-up dialoog.

Open de DDMS Perspective > File Explorer, daar vind je de opgegeven data terug in de map \data\data\...\shared\_prefs.

(\* dit laatste lukte mij niet)

SAMENVATTING van dit hoofdstuk: zie tabel p 216-217

**Wrox Hoofdstuk 5 : Afbeeldingen en Menu’s**

In dit hoofdstuk leer je werken met :

* Gallery (afbeeldingengalerij)
* ImageSwitcher
* GridView
* ImageView
* Option Menus en Context Menus.
* Tijd weergeven met AnalogClock en DigitalClock views.
* Web content weergeven met een WebView view

Een afbeelding zegt meer dan 1000 woorden , zegt men toch … dus moet je in een app ook afbeeldingen kunnen weergeven.

**Gallery en ImageView views (p 220)**

Een Gallery view toont afbeeldingen in een soort “filmstrip” weergave.

Een ImageView toont 1 afbeelding uit de reeks, in een groter formaat.

Beide views worden gebruikt in het voorbeeld **Gallery**.

Er komt ook een nieuw xml bestand bij kijken : **attrs.xml** in de map \res\values, hierin wordt genoteerd waar de array met afbeeldingen te vinden is.

Om in een Gallery een aantal afbeeldingen te kunnen weergeven wordt een **Adapter klasse** gebruikt. Dit is een systeem om een set **data** (uit een array, een database, enz) te **binden**  aan een V**iew.** De Adapterklasse creëert bovendien ook de View, zie de getView() methode hierin, deze levert een ImageView op met daarin de gewenste afbeelding.

Android levert verschillende dergelijke “native” Adapterklassen, zodat je enkel je eigen aanpassing daarvan moet schrijven , bvb hier de class ImageAdapter extends BaseAdapter.

**ImageSwitcher (p 226)**

Een verbeterde versie van vorig programma, waarbij een “overgangseffect” gebruikt wordt tussen de afbeeldingen. Zie voorbeeld **ImageSwitcher**.

Let bij de java klasse op de extra … implements ViewFactory en de imageSwitcher opdrachten.

In de code vind je twee voorbeeld-effecten: fade in/out en slide in/out (wijzig de comments).

**GridView (p 231)**

Nog een andere View waarmee je (onder andere) afbeeldingen kan tonen.

Niet te verwarren met **GridLayout !**

Zie voorbeeld **Grid.**

Dit programma toont enkel welke afbeelding je aangeklikt hebt.

Oefening : breid dit programma uit zodat de afbeelding ook groter in een ImageView verschijnt (zie oplossing achteraan deze tekst (met dank aan Sam Breunig !).

**Menu’s (p 234)**

Twee soorten menu’s worden besproken :  **Options Menu** en **Context Menu.**

Een Options Menu wordt zichtbaar via de “menu” knop (een hardware knop in de emulator of op de device, of als je geen menu-knop hebt, een softbutton : )

Een Context Menu wordt geactiveerd door lang op een View te duwen.

Voor je met menu’s kan werken moet je nog de invulling doen van wat in het menu zal verschijnen, en ook wat er moet gebeuren als een gebruiker een of andere optie kiest.

Dit doe je door eerst twee **Helper** methoden te maken : CreateMenu() en MenuChoice().

Zie voorbeeld **Menus** **(p 235).**

**Options Menu (p 238)**

Hiertoe moet je nog twee methodes bijvoegen : onCreateOptionsMenu() , hiermee wordt het menu opgebouwd, en onOptionsItemSelected(), waarmee de betreffende keuze afgehandeld wordt.

**Context Menu ( p240)**

Dit menu wordt geassocieerd met een bepaalde View, bvb. een knop of een afbeelding. De koppeling met de View doe je door de methode setOnCreateContextMenuListener() toe te passen op de View.

Zie uitbreiding van voorbeeld **Menus** **(p 240)**, waar dit voor een Button View getoond wordt. De methode onCreateContextMenu() in het hoofdprogramma wordt opgeroepen als de button lang aangeklikt wordt.

Oefening : je krijgt in de demo de zelfde menu te zien als bij Options Menu. Pas het programma aan zodat een ander menu verschijnt, aangepast aan de knop.

**Nog enkele nuttige views : AnalogClock, DigitalClock en WebView (p 242)**

**AnalogClock** en **DigitalClock**

Dit zijn twee eenvoudige Views die je enkel in xml moet declareren.

Ze kunnen enkel de lokale tijd weergeven.

**WebView**

Hiermee kan je binnen je eigen activity een webbrowser gebruiken.

Zie voorbeeld **WebView** (p 243).

Hierin maak je een WebView aan, en met loadUrl() kan je een pagina in de WebView laden.

Merk op dat je in het manifest bestand een permissie moet toevoegen om op internet te geraken: <uses-permission android:name=”android.permission.INTERNET”/>

Een probleem ontstaat als de pagina je wil “re-directen”.

In dit geval moet je een subklasse van de klasse WebViewClient maken, waarin je de methode shouldOverrideUrlLoading() moet overriden.

In de demo (p 247) staat verder nog hoe een (kleine) html-pagina in je source code kan “ingebakken” worden, ofwel geladen vanuit de folder **“\assets”**

SAMENVATTING van dit hoofdstuk: zie tabel p 250.

**Wrox H6: Data Persistence (p 251)**

Hierin wordt aangeleerd hoe je data blijvend kan bewaren op drie manieren :

* Kleine hoeveelheden data bewaren in een SharedPreferences object
* Data schrijven en lezen in klassieke bestanden
* Een SQLite database opzetten

**SharedPreferences (p251)**

Dit is een object waarin de gebruiker zijn voorkeuren (preferences) kan bewaren, bvb. het ingestelde lettertype (Font), dit hoef je dan niet telkens opnieuw aan te passen als je de applicatie gebruikt.De info wordt ook hier opgeslagen als “name/value” koppels, het systeem plaatst deze in een **xml** bestand myapppreferences.xml.

Bij het starten van de app worden de preferences uit die xml geladen met de opdracht

addPreferencesFromResource(R.xml.myapppreferences)

Je moet hiervoor wel een aparte opstartklasse maken … extends PreferenceActivity

Zie voorbeeld **UsingPreferences** (p252).

In een tweede voorbeeld wordt getoond hoe je deze preferences vanuit java code kan benaderen (p 259): zowel het opvragen als het oproepen van een Editor om de preferences te wijzigen. In de methode onClickDisplay() maak je eerst een instantie van SharedPreferences met de methode getSharedPreferences() , met de methode getString() kan je dan bvb. de value ophalen die bij de naam “editTextPref” hoort.

In de methode onClickModify() maak je een SharedPreferences.Editor aan, waarvan je met methode putString() een waarde kan wijzigen.

Op p 261 vind je nog hoe je de xml een andere naam kan geven dan de standaard naam.

**FILES (p 263)**

Dit is een universeel toepasbare techniek om bestanden te schrijven en te lezen.

Android gebruikt gelukkig de klassen uit de package java.io, zodat je hiermee allicht niet zoveel moeite zal hebben.

Zie voorbeeld **Files**.

De Save en Load code op p 265 vertoont grote gelijkenissen met wat we eerder zagen in Java, al gebruikt men hier een OutputStreamWriter i.p.v. BufferedWriter, characters worden hierin tot bytes omgevormd. OutputStreamWriter bevat zelf ook de nodige buffer.

De bestanden van dit voorbeeld komen terecht op de interne opslag (flash geheugen in de device).

**Externe opslag (p 268)**

Om op de externe opslag (SD card) te schrijven, moet je iets meer code toevoegen, zie p 268-269.

**Static Resources (p 272)**

Soms wil je gewoon ook wat “files” in je app meegeven voor gebruik tijdens de run-fase, bvb. helpfiles, lijstje met afkortingen, foutmeldingen, enz.

Deze plaats je in een map **\res\raw** (moet je zelf aanmaken).

Op p 273 vind je de nodige code om dergelijke file te openen, raar genoeg deze keer wel met BufferedReader !

**DATABASES (p 273)**

Dit is de “strafste” methode van data-opslag, maar meteen ook de meest veelzijdige.

Wie het vak databases niet kreeg kan hier een inleiding vinden over het gebruikte SQLite database systeem dat in Android gebruikt wordt: <http://zetcode.com/db/sqlite/introduction/>.

Of als je liever een filmpje kijkt … <http://www.youtube.com/watch?v=f428dSRkTs4> of dit hier <http://mrbool.com/a-brief-introduction-to-sqlite/10623> (voor linux fans).

Zie voorbeeld **Databases** (p 274).

Eerst wordt een helper klasse DBAdapter gemaakt, die alle complexiteit van de database in zich heeft. Voor de oproepende code wordt het dan iets eenvoudiger om de data te gebruiken.

De voorbeeld-database heeft per **record** drie **velden** : \_*id, naam en e-mail*.

De klasse bevat alle nodige constanten en een hele reeks methoden:

* de database aanmaken, upgraden, openen, sluiten
* alle records opvragen
* records bijvoegen, deleten, wijzigen.

Typische opdrachten hebben allen de vorm :

db.open();

db.insertContact(“Mary Jackson”, “mary@jackson.com”)

db.getContact(2);

db.getAllContacts()

db.deleteContact(1)

**Vooraf gemaakte database (p 285)**

Met de grafische tool **SQLite Database Browser** kan je een database visueel opmaken, om dan in een app te gebruiken. Je kan deze tool hier gratis downloaden :

<http://sourceforge.net/projects/sqlitebrowser/>

De vooraf gemaakte database plaats je in de map \assets , let wel op dat de bestandsnaam enkel lowercase karakters mag bevatten, bvb. ***mydb*** .

Verdere code, zie p 286-289

SAMENVATTING van dit hoofdstuk: zie tabel p 291.

**Wrox H7 : Content Providers (p 293)**

In dit hoofdstuk wordt aangeleerd hoe je data deelt met andere apps.

De techniek maakt gebruik van zogenaamde **Content Providers.**

Er zijn ingebouwde content providers in Android, en je kan ook zelf content providers aanmaken.

De ingebouwde content providers zijn :

**Browser** : geschiedenis van de browser, de bookmarks, enz

**CallLog** : gegevens i.v.m. je telefoonoproepen

**Contacts**: je lijst van contactpersonen raadplegen

**MediaStore** : audio, video en afbeeldingen

**Settings** : de settings van de device

Al deze gegevens kan je raadplegen, en in sommige gevallen ook aanpassen.

Hoe het systeem de data opslaat hoef je niet te weten, dit kan in een bestand zijn, in een database, of zelfs online (zie H6).

Om data op te vragen moet je enkel een “query string” maken (zoals bij SQL), in dit geval in de vorm van een **URI:**

<standard\_prefix>://<authority>/<data\_path>/<id>

* Een Uniform Resource Identifier (URI), is een internet-protocolelement, gebaseerd op eerdere voorstellen van Tim Berners-Lee. Het is een eenmalige benaming van een "bron", een stuk informatie, data of dergelijke.  
  bron: <http://nl.wikipedia.org/wiki/Uniform_Resource_Identifier>

Betekenis van de onderdelen :

<standard\_prefix>:// is steeds : **content://**

<authority>/ vb: **contacts** , ofwel een volledige naam zoals

**be.vives.provider**

<data\_path>/ de naam van de data die je wil , bvb. **people**, uit de contacts

<id> optionele identifier als je een bepaald record zoekt, vb **2**

Voorbeeld : om de tweede contactpersoon uit Contacts op te vragen is de volledige URI:

content://contacts/people/2

Andere voorbeelden:

**Query string Resultaat**

content://media/internal/images Returns a list of all the internal images on the device

content://media/external/images Returns a list of all the images stored on the external  
 storage (e.g., SD card) on the device

content://call\_log/calls Returns a list of all calls registered in the Call Log

content://browser/bookmarks Returns a list of bookmarks stored in the browser

Zie voorbeeld **Provider** voor het praktische gebruik **(p 295)**.

De contactenlijst wordt geladen door een opdracht waarin de URI gebruikt wordt, zie code:

Uri allContacts = Uri.parse(“content://contacts/people”);

Met de constructor van de CursorLoader klasse wordt dan een Cursor gevuld met de data :

CursorLoader cursorLoader = new CursorLoader(

this,

allContacts,

null,

null,

null ,

null);

c = cursorLoader.loadInBackground();

Een **Cursor** is niets anders dan een tweedimensionele tabel die de data bevat die je na een query terugkrijgt. De tabel wordt gevuld in een *background thread*  zodat je activity niet geblokkeerd geraakt.

In dit voorbeeld wordt de data tot slot via een **SimpleCursorAdapter** object omgevormd tot een array van TextViews, met daarin de gevraagde data.

Opdat heel dit systeem zou werken moet in de AndroidManifest.xml een   
READ\_CONTACTS permission opgenomen zijn.

**Voorgedefinieerde Query Strings (p 300)**

Android kent enkel voorgeprogrammeerde strings die men toch veel gebruikt, vb:

*ContactsContract.Contacts.CONTENT\_URI*

*Browser.BOOKMARKS\_URI*

*Browser.SEARCHES\_URI*

*CallLog.CONTENT\_URI*

*MediaStore.Images.Media.INTERNAL\_CONTENT\_URI*

*MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI*

*Settings.CONTENT\_URI*

Dan hoef je zelf de URI niet ineen te knutselen …

De lijst afdrukken in het LogCat venster : zie p 301.

Ook de telefonnummers opvragen van de contactpersonen is nog een stapje ingewikkelder, zie p 302.

**Projections**

Dit slaat op het aantal kolommen dat je na een query wil terugkrijgen, zie p 303

Daartoe moet je zelf eerst een array maken, voorzien van de nodige kolommen, voor bvb. ID, NAME en NUMBER. Deze array wordt dan als derde argument aan de CursorLoader constructor meegegeven.

**Filtering (p 304)**

Het vierde argument in die constructor kan een filter zijn, bvb :

…, ContactsContract.Contacts.DISPLAY\_NAME + “ LIKE ‘%Lee’”, …

Waarmee je enkel de namen krijgt die eindigen op ‘Lee’ (de syntax volgt de SQL regels).

Je kan dit argument ook nog splitsen in twee stukken, zoals hier (4e + 5e argument):

…, ContactsContract.Contacts.DISPLAY\_NAME + “ LIKE ?”,  
 new String[] {“%Lee”}, …

**Sorting (p 305)**

Het zesde en laatste argument van de CursorLoader constructor. Om bvb. de namen in oplopende volgorde te sorteren geef je dit argument op :

…, ContactsContract.Contacts.DISPLAY\_NAME + “ ASC”);

**Eigen Content Provider maken (p 305)**

Zie voorbeeld **ContentProviders** waarin een boekenlijst opgeslagen wordt.

Dit is vrij eenvoudig : je moet een subklasse maken van de klasse ContentProvider, en enkele methoden *overriden*, zie p 306-309.

**De eigen Content Provider gebruiken (p 314)**

Het voorbeeld laat toe een boek toe te voegen aan de content provider, en ook een lijst op te vragen van de opgeslagen boeken.

SAMENVATTING van dit hoofdstuk: zie tabel p 320.

**Aanvulling : tips & trucs**

(alle input voor deze rubriek is welkom !)

**Instellen hoe je “input” moet verlopen**: welk keyboard (tekst of getallen) , met of zonder spellingscorrectie, suggestie doen voor automisch aanvullen, eerste karakter automatisch een hoofdletter, enz …

Zie deze uitleg : <http://developer.android.com/training/keyboard-input/style.html>

**Problemen met de AVD : device wil niet starten of blijft hangen :**

Mogelijke oplossingen :

* Zet de RAM niet te hoog, 512 MB moet volstaan
* Vink de optie “snapshot” aan (werkt soms averechts, bij mij ging het dan niet meer)

**Real Device**

* Bij problemen : device losmaken, Eclipse herstarten, device terug aansluiten

**DDMS (Dalvik Debug Monitor Server)**

Om een bestand vanop je device te kunnen inspecteren moet je “extract from device” uitvoeren. Zorg daarbij dat in het linker deel je device aangeklikt is.

**Verkeerde app wordt gestart, of je app start niet ?**

Rechtsklikken op je project en Run As > Run Configurations : daar controleren of de naam van je actieve project ingevuld is, desnoods aanpassen.

**Zoomknoppen doen verschijnen in de emulator**

Kl;ik en sleep over de afbeelding

**Keyboard oproepen**

Hoe roep je het keyboard op als het niet vanzelf verschijnt ?   
Bij sommige toestellen lang op de MENU key drukken, maar als je geen MENU key hebt … 🡪 installeer Hacker’s Keyboard, en pas daarvan de instellingen aan zodat een permanente lijn in de notification drawer staat waarmee je het keyboard kan oproepen.

ECLIPSE : alle ontbrekende **imports** “fixen” : **Ctrl-Shift-O** (Organize Imports)

ECLIPSE : code herformatteren : klik rechts op Project , dan: Source > Format

**Hyperlinks**

<http://www.iz5vtp.info/ham-radio-android-apps>

<http://www.androidplanet.nl/>

<https://www.youtube.com/watch?v=b3LWF4xa6nE>

<http://www.wolphi.com/interface>

<http://www.youtube.com/watch?v=ltadM2qRmoM&feature=em-hot-vrecs>

<http://www.digitaltrends.com/mobile/google-nexus-7-problems/>

<http://www.youtube.com/watch?v=sYGovjtQCWM>

<http://www.wolphi.com/wolphilink/wolphilinkmanual.pdf>

<http://www.yo3ggx.ro/FT8x7_DIY_Bluetootth_CAT_interface_v1.pdf>

<http://www.computeridee.nl/artikel/android/apps-maken-voor-android>

<http://developer.android.com>

Naast de pure technische code, moet een app ook “aantrekkelijk” zijn voor de gebruiker. Op de Android Developers site spreekt men van drie vereisten voor een goeie app :

Enchant me, Simplify my life , Make me amazing. Lees hier wat ze daarmee bedoelen :  
<http://developer.android.com/design/get-started/principles.html>

<http://developer.android.com/design/get-started/principles.html>

<http://androidworld.nl/nieuws/devtutorial-1-ontwikkelen-voor-android-de-benodigde-tools/>

<http://androidworld.nl/nieuws/devtutorial-1-ontwikkelen-voor-android-de-benodigde-tools/>

<http://telescript.denayer.wenk.be/~w1/Een-applicatie-schrijven.shtml>

zelfstudie sites:

<http://www.androidplanet.nl/33650/leren-ontwikkelen-voor-android-hier-200-gratis-videotutorials/>

<http://www.zelfstudie.be/cursus-app-maken-voor-android.php>

Manieren om te tekenen :

op een View zelf, of in een Bitmap op een Canvas : lees dit :

<http://developer.android.com/guide/topics/graphics/2d-graphics.html>

Toast die korter is dan de standaard Toast.LENGTH\_SHORT (die 2,5 seconden duurt) ?

(nog eens te testen):

final Toast toast = Toast.makeText(ctx, "This message will disappear in 1 second", Toast.LENGTH\_SHORT);

toast.show();

Handler handler = new Handler();

handler.postDelayed(new Runnable() {

@Override

public void run() {

toast.cancel();

}

}, 1000);

bron : <http://stackoverflow.com/questions/3775074/set-toast-appear-length>

**Oplossing van om het voorbeeld Grid (blz 19 in deze tekst)** ook de aangeklikte afbeelding te laten tonen.

**In main.xml**

wijzig de layout in FrameLayout en voeg een ImageView toe vóór de bestaande GridView :

<ImageView

android:id="@+id/imageView1"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

**In GridActivity.java**

* Declaratie bijvoegen : ImageView image1;
* In onCreate(), na setContentView :  
   image1 = (ImageView) findViewById(R.id.imageView1);
* Juist vóór de Toast opdracht in onItemClick() dit toevoegen :

image1.setVisibility(0);

image1.setImageResource(imageIDs[position]);

image1.bringToFront();

(de Toast kan je desgewenst ook weglaten)

**TEKENEN**

1e methode

(bron : <http://stackoverflow.com/questions/3616676/how-to-draw-a-line-in-android> )

Maak een aparte klasse in bestand DrawView.java met volgende inhoud :

public class DrawView extends View {

Paint paint = new Paint();

public DrawView(Context context) {

super(context);

paint.setColor(Color.BLACK);

}

@Override

public void onDraw(Canvas canvas) {

canvas.drawLine(0, 0, 20, 20, paint);

canvas.drawLine(20, 0, 0, 20, paint);

}

}

In het hoofdprogramma voeg je bij :

DrawView drawView;

…

En vervang je onCreate() door dit :

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

drawView = new DrawView(this);

drawView.setBackgroundColor(Color.WHITE);

setContentView(drawView);

}

Tekenen met SurfaceView

Filmpje : <http://www.youtube.com/watch?v=Z2YogvILjvo>

Lijken goede tutorials te zijn … eens allemaal bekijken ?

<http://www.youtube.com/user/mybringback/videos>