



# Android Development - Tutorial

**Based on Android 4.4**

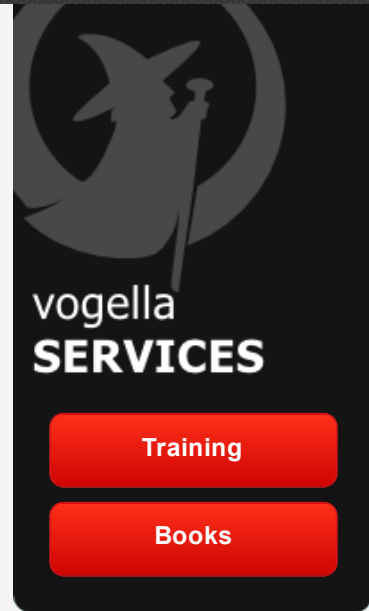
**Lars Vogel**

Version 11.8

Copyright © 2009, 2010, 2011, 2012, 2013, 2014 Lars Vogel

12.05.2014

## Revision History





## Development with Android and Eclipse

This tutorial describes how to create Android applications. It primarily uses the Eclipse IDE for development. It is based on Android 4.4 (KitKat).

### Table of Contents

#### 1. What is Android?

- 1.1. The Android operating system
- 1.2. Task
- 1.3. Android platform components
- 1.4. Google Play

#### 2. Android Development Tools

- 2.1. Android SDK
- 2.2. Android debug bridge (adb)
- 2.3. Android Developer Tools and Android Studio
- 2.4. Dalvik Virtual Machine
- 2.5. Just in time compiler on Dalvik
- 2.6. Android RunTime (ART)
- 2.7. How to develop Android applications
- 2.8. Conversion process from source code to Android application

#### 3. Security and permissions

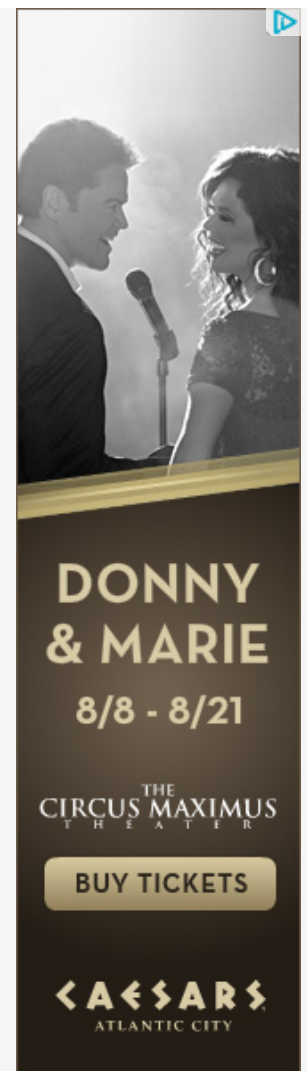
- 3.1. Security concept in Android
- 3.2. Permission concept in Android

#### 4. Installation

- 4.1. Install Android Developer Tools
- 4.2. Other Eclipse installation options

#### 5. Android device emulator and Android Virtual Devices

- 5.1. Android emulator and Android Virtual Device



- 5.2. Android device emulator shortcuts**
- 5.3. Google vs. Android AVD**
- 5.4. Speed optimization**
- 5.5. Intel system image**
- 5.6. Alternative emulator**

## **6. Exercise: Create an Android Virtual Device (AVD)**

- 6.1. Target**
- 6.2. Create an AVD**
- 6.3. Start your AVD**

## **7. Exercise: Create an Android application in Eclipse**

- 7.1. Using the Android project wizard**
- 7.2. Create an Android project**

## **8. Exercise: Start the generated Android application**

- 8.1. Start the AVD**
- 8.2. Start the application**

## **9. Solving Android development problems**

## **10. Connect with the Android source code**

- 10.1. Connect the source to your Android JAR file**
- 10.2. Validate**

## **11. Integration of ADT into Eclipse**

- 11.1. Android integration into the Java perspective**
- 11.2. Android wizards**

## **12. DDMS perspective**

- 12.1. Android perspective**
- 12.2. Emulator Control**
- 12.3. File explorer**

## **13. Parts of a Android application**

- 13.1. Android application**
- 13.2. Android software components**

### **13.3. Context**

## **14. Android application components overview**

### **14.1. Activity**

### **14.2. BroadcastReceiver**

### **14.3. Service**

### **14.4. ContentProvider**

## **15. Base user interface components in Android**

### **15.1. Activity**

### **15.2. Fragments**

### **15.3. Views and layout manager**

### **15.4. Device configuration specific layouts**

## **16. Other important Android elements**

### **16.1. Home screen and lock screen widgets**

### **16.2. Live Wallpapers**

## **17. The Android manifest**

### **17.1. Configuration of your Android application**

### **17.2. Declaring components in the manifest file**

### **17.3. Permissions**

### **17.4. AndroidManifest.xml example file**

## **18. The Android manifest**

### **18.1. Version and package**

### **18.2. Application and components**

### **18.3. Minimum and target SDK**

### **18.4. Permissions**

### **18.5. Required device configuration**

### **18.6. Installation location**

### **18.7. More info**

## **19. Resources**

### **19.1. Resource files**

### **19.2. Resource example**

### **19.3. Resource qualifiers**

### **19.4. Resource IDs and R.java**

### **19.5. Good practices for resources IDs**

### **19.6. System resources**

## **20. Layout resource files**

### **20.1. Activities and layouts**

### **20.2. XML layout files**

### **20.3. Defining IDs**

### **20.4. Good practice: Predefined IDs via a separate file**

### **20.5. Performance considerations with layouts**

## **21. Views**

### **21.1. View class**

### **21.2. Standard views**

### **21.3. Custom views**

## **22. Layout Manager and ViewGroups**

### **22.1. What is a layout manager?**

### **22.2. Important layout managers**

### **22.3. Layout attributes**

### **22.4. FrameLayout**

### **22.5. LinearLayout**

### **22.6. RelativeLayout**

### **22.7. GridLayout**

### **22.8. ScrollView**

## **23. Exercise: Use layouts and add interaction**

### **23.1. Review layout**

### **23.2. Remove exiting views**

### **23.3. Add and configure views**

### **23.4. Add interactive button**

### **23.5. Validate button interaction**

### **23.6. Display text from your EditText field**

### **23.7. Validate popup message**

### **23.8. Solution**

## **24. Exercise: Influence view layout at runtime**

### **24.1. Add radio group and radio buttons to your layout**

### **24.2. Change radio group orientation dynamically**

### **24.3. Validate**

## **25. Exercise: Create a temperature converter**

- 25.1. Demo application**
- 25.2. Create Project**
- 25.3. Create attributes**
- 25.4. Using the layout editor**
- 25.5. Add views to your layout file**
- 25.6. Edit view properties**
- 25.7. Create utility class**
- 25.8. Change the activity code**
- 25.9. Start application**

## **26. Using Resources**

- 26.1. References to resources in code**
- 26.2. Accessing views from the layout in an activity**
- 26.3. Reference to resources in XML files**
- 26.4. Reference to Android system resources in XML files**

## **27. Assets**

- 27.1. Whats are assets?**
- 27.2. Accessing assets**

## **28. Exercise: Using resources in XML files and in code**

- 28.1. Add images to your project**
- 28.2. Add views to your project**
- 28.3. Assign image to your image view**
- 28.4. Replace images via button click**
- 28.5. Validate**

## **29. Exercise: Using ScrollView**

## **30. Deployment**

- 30.1. Overview**
- 30.2. Deployment via Eclipse**
- 30.3. Export your application**
- 30.4. Via external sources**
- 30.5. Google Play (Market)**

## **31. Support free vogella tutorials**

- 31.1. Thank you**

### 31.2. Questions and Discussion

### 32. Links and Literature

#### 32.1. Source Code

#### 32.2. Android Online Resources

#### 32.3. vogella Resources

## Dropbox for Business

 [dropbox.com/Business](https://dropbox.com/Business)

Share Files With Your Business. Easy Cloud Backup. Free Trial.



# 1. What is Android?

## 1.1. The Android operating system

*Android* is an operating system based on the Linux kernel. The project responsible for developing the Android system is called the *Android Open Source Project* (AOSP) and is primarily lead by Google.

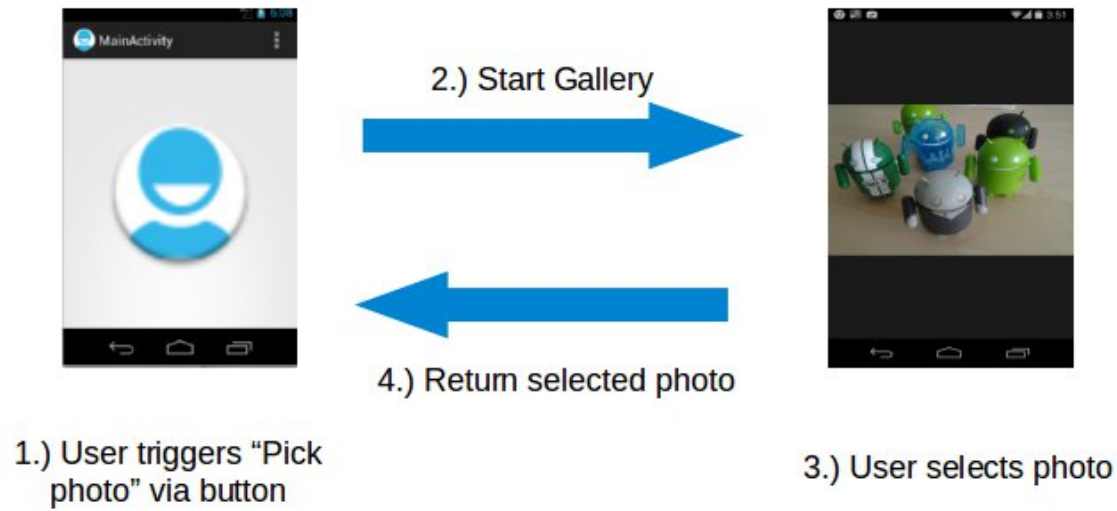
The Android system supports background processing, provides a rich user interface library, supports 2-D and 3-D graphics using the OpenGL-ES (short OpenGL) standard and grants access to the file system as well as an embedded SQLite database.

An Android application typically consists of different visual and non visual components and can reuse components of other applications.

## 1.2. Task

In Android the reuse of other application components is a concept known as *task*. An application can access other Android components to achieve a task. For example, from a component of your application you can trigger another component in the Android system, which manages photos, even if this component is not part of your application. In this component you select a photo and return to your application to use the selected photo.

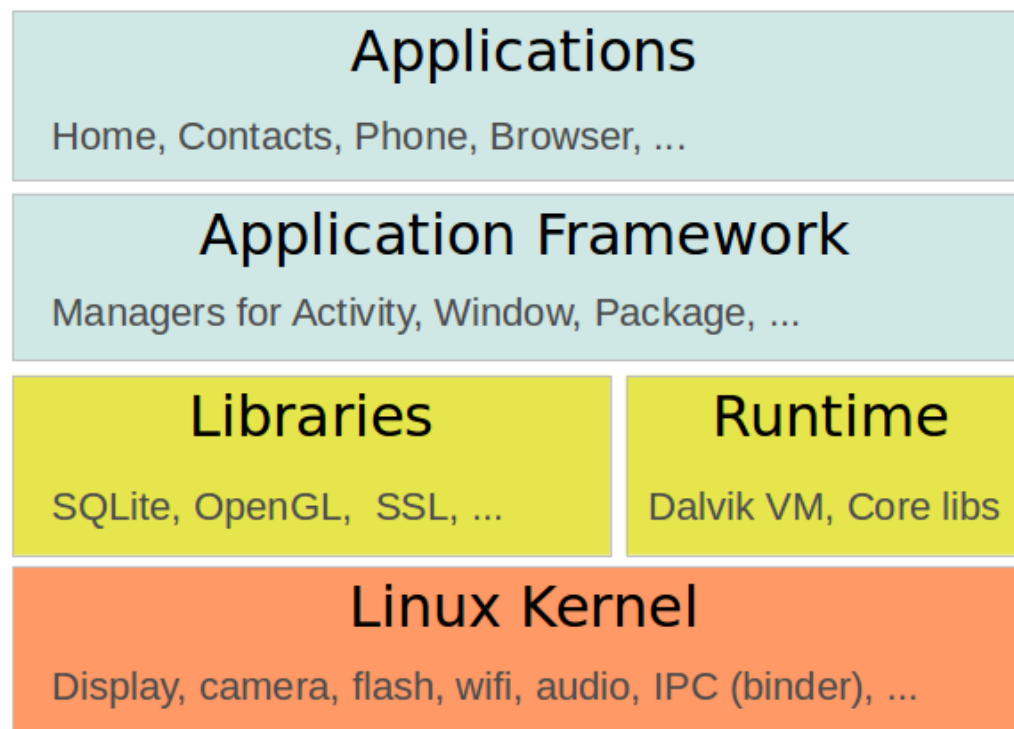
Such a flow of events is depicted in the following graphic.



### 1.3. Android platform components

The Android system is a full software stack, which is typically divided into the four areas as depicted in the following graphic.





The levels can be described as:

- Applications - The Android Open Source Project contains several default application, like the Browser, Camera, Gallery, Music, Phone and more.
- Application framework - An API which allows high-level interactions with the Android system from Android applications.
- Libraries and runtime - The libraries for many common functions (e.g.: graphic rendering, data storage, web browsing, etc.) of the Application Framework and the Dalvik runtime, as well as the core Java libraries for running Android applications.
- Linux kernel - Communication layer for the underlying hardware.

The Linux kernel, the libraries and the runtime are encapsulated by the application framework. The Android application developer typically works with the two layers on top to create new Android applications.

## 1.4. Google Play

Google offers the *Google Play* service, a marketplace in which programmers can offer their Android applications to

Android users. Customers use the *Google Play* application which allows them to buy and install applications from the Google Play service.

Google Play also offers an update service. If a programmer uploads a new version of his application to Google Play, this service notifies existing users that an update is available and allows them to install the update.

Google Play provides access to services and libraries for Android application programmers, too. For example, it provides a service to use and display Google Maps and another to synchronize the application state between different Android installations. Providing these services via Google Play has the advantage that they are available for older Android releases and can be updated by Google without the need for an update of the Android release on the phone.

## 2. Android Development Tools

### 2.1. Android SDK

The *Android Software Development Kit* (Android SDK) contains the necessary tools to create, compile and package Android applications. Most of these tools are command line based. The primary way to develop Android applications is based on the Java programming language.

### 2.2. Android debug bridge (adb)

The Android SDK contains the *Android debug bridge* (adb), which is a tool that allows you to connect to a virtual or real Android device, for the purpose of managing the device or debugging your application.

### 2.3. Android Developer Tools and Android Studio

Google provides two integrated development environments (IDEs) to develop new applications.

The *Android Developer Tools* (ADT) are based on the Eclipse IDE. ADT is a set of components (plug-ins), which extend the Eclipse IDE with Android development capabilities.

Google also supports an IDE called *Android Studio* for creating Android applications. This IDE is based on the IntelliJ IDE.

Both IDEs contain all required functionality to create, compile, debug and deploy Android applications. They also allow the developer to create and start virtual Android devices for testing.

Both tools provide specialized editors for Android specific files. Most of Android's configuration files are based on XML. In this case these editors allow you to switch between the XML representation of the file and a structured user interface for entering the data.

## 2.4. Dalvik Virtual Machine

The Android system uses a special virtual machine, i.e., the *Dalvik Virtual Machine* (Dalvik) to run Java based applications. Dalvik uses a custom bytecode format which is different from Java bytecode.

Therefore you cannot run Java class files on Android directly; they need to be converted into the Dalvik bytecode format.

## 2.5. Just in time compiler on Dalvik

Similar to the JVM, Dalvik optimizes the application at runtime. This is known as *Just In Time* (JIT) compilation. If a part of the application is called frequently, Dalvik will optimize this part of the code and compile it into machine code which executes much faster.

## 2.6. Android RunTime (ART)

With Android 4.4, Google introduced the *Android RunTime* (ART) as optional runtime for Android 4.4. It is expected that versions after 4.4 will use ART as default runtime.

ART uses *Ahead Of Time* compilation. During the deployment process of an application on an Android device, the application code is translated into machine code. This results in approx. 30% larger compile code, but allows faster execution from the beginning of the application.

## 2.7. How to develop Android applications

Android applications are primarily written in the Java programming language.

During development the developer creates the Android specific configuration files and writes the application logic in the Java programming language.

The ADT or the Android Studio tools convert these application files, transparently to the user, into an Android application. When developers trigger the deployment in their IDE, the whole Android application is compiled, packaged, deployed and started.

## 2.8. Conversion process from source code to Android application

The Java source files are converted to Java class files by the Java compiler.

The Android SDK contains a tool called *dx* which converts Java class files into a *.dex* (Dalvik Executable) file. All class files of the application are placed in this *.dex* file. During this conversion process redundant information in the class files are optimized in the *.dex* file.

For example, if the same `String` is found in different class files, the `.dex` file contains only one reference of this `String`.

These `.dex` files are therefore much smaller in size than the corresponding class files.

The `.dex` file and the resources of an Android project, e.g., the images and XML files, are packed into an `.apk` (Android Package) file. The program *aapt* (Android Asset Packaging Tool) performs this step.

The resulting `.apk` file contains all necessary data to run the Android application and can be deployed to an Android device via the *adb* tool.

## 3. Security and permissions

### 3.1. Security concept in Android

The Android system installs every Android application with a unique user and group ID. Each application file is private to this generated user, e.g., other applications cannot access these files. In addition each Android application is started in its own process.

Therefore, by means of the underlying Linux kernel, every Android application is isolated from other running applications.

If data should be shared, the application must do this explicitly via an Android component which handles the sharing of the data, e.g., via a service or a content provider.

### 3.2. Permission concept in Android

Android contains a permission system and predefines permissions for certain tasks. Every application can request required permissions and also define new permissions. For example, an application may declare that it requires access to the Internet.

Permissions have different levels. Some permissions are automatically granted by the Android system, some are automatically rejected. In most cases the requested permissions are presented to the user before installing the application. The user needs to decide if these permissions shall be given to the application.

If the user denies a required permission, the related application cannot be installed. The check of the permission is only performed during installation, permissions cannot be denied or granted after the installation.

An Android application declares the required permissions in its *AndroidManifest.xml* configuration file. It can also define additional permissions which it can use to restrict access to certain components.





### Note

Not all users pay attention to the required permissions during installation. But some users do and they write negative reviews on Google Play if they believe the application is too invasive.

## 4. Installation

### 4.1. Install Android Developer Tools

#### 4.1.1. Download packaged Android Developer Tools

Google provides a packaged and configured Android development environment based on the Eclipse IDE called *Android Developer Tools*. Under the following URL you find an archive file which includes all required tools for Android development: [Getting the Android SDK](#).

#### 4.1.2. Stand-alone ADT installation

Extract the zip file and start the Android Developer Tools (Eclipse) which are located in the *eclipse* folder. You can do this by double-clicking on the *eclipse* native launcher (e.g., *eclipse.exe* under Windows).

#### 4.1.3. Update an existing Eclipse IDE

See [???](#) for a description on how to update your existing Eclipse IDE to perform Android development.

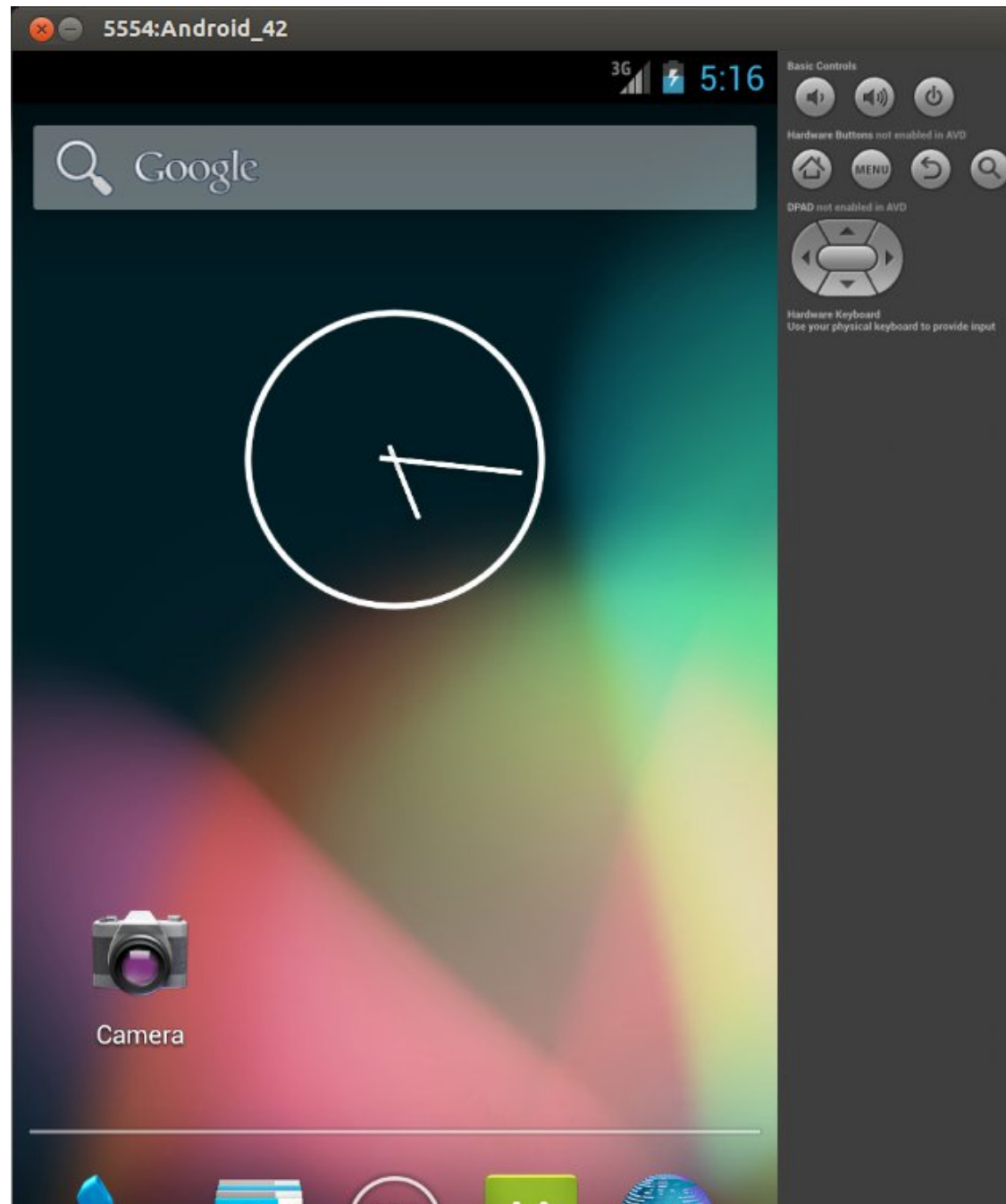
### 4.2. Other Eclipse installation options

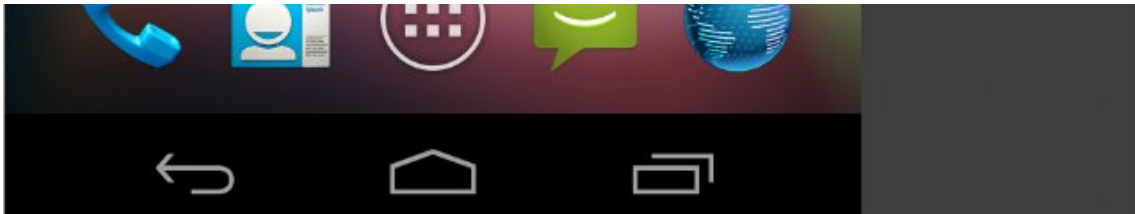
The simplest way to start Android development with Eclipse is to download a complete and pre-configured bundle as described in [Section 4.1.1, “Download packaged Android Developer Tools”](#). It is also possible to update an existing Eclipse installation. Please see [Android installation](#) for a detailed description

## 5. Android device emulator and Android Virtual Devices

### 5.1. Android emulator and Android Virtual Device

The Android SDK contains an Android device emulator. This emulator can be used to run an *Android Virtual Device* (AVD), which emulates a real Android phone. Such an emulator is displayed in the following screenshot.





AVDs allow you to test your Android applications on different Android versions and configurations without access to the real hardware.

During the creation of your AVD you define the configuration for the virtual device. This includes, for example, the resolution, the Android API version and the density of your display.

You can define multiple AVDs with different configurations and start them in parallel. This allows you to test different device configurations at once.

### 5.2. Android device emulator shortcuts

The following table lists useful shortcuts for working with an AVD.

**Table 1. Android device emulator shortcuts**

Shortcut	Description
<b>Alt+Enter</b>	Maximizes the emulator.
<b>Ctrl+F11</b>	Changes the orientation of the emulator from landscape to portrait and vice versa.
<b>F8</b>	Turns the network on and off.

### 5.3. Google vs. Android AVD

During the creation of an AVD you decide if you want to create an Android device or a Google device.

An AVD created for Android contains the programs from the *Android Open Source Project*. An AVD created for the Google API's contains additional Google specific code.

AVDs created for the Google API allow you to test applications which use Google Play services, e.g., the new Google maps API or the new location services.

## 5.4. Speed optimization

During the creation of an emulator you can choose if you either want *Snapshot* or *Use Host GPU* enabled.



### Note

The dialog implies that you can select both options, but if you do, you get an error message that these options can not be selected together.

If you select the *Snapshot* option, the second time you start the device it is started very fast, because the AVD stores its state if you close it. If you select *Use Host GPU* the AVD uses the graphics card of your host computer directly which makes the rendering on the emulated device much faster.



**Create new Android Virtual Device (AVD)**

AVD Name:

Device:

Target:

CPU/ABI:

Keyboard: ☒ Hardware keyboard present

Skin: ☒ Display a skin with hardware controls

Front Camera:

Back Camera:

Memory Options: RAM:  VM Heap:

Internal Storage:

SD Card:

☒ Size:

☐ File:

Emulation Options: ☒ Snapshot ☐ Use Host GPU

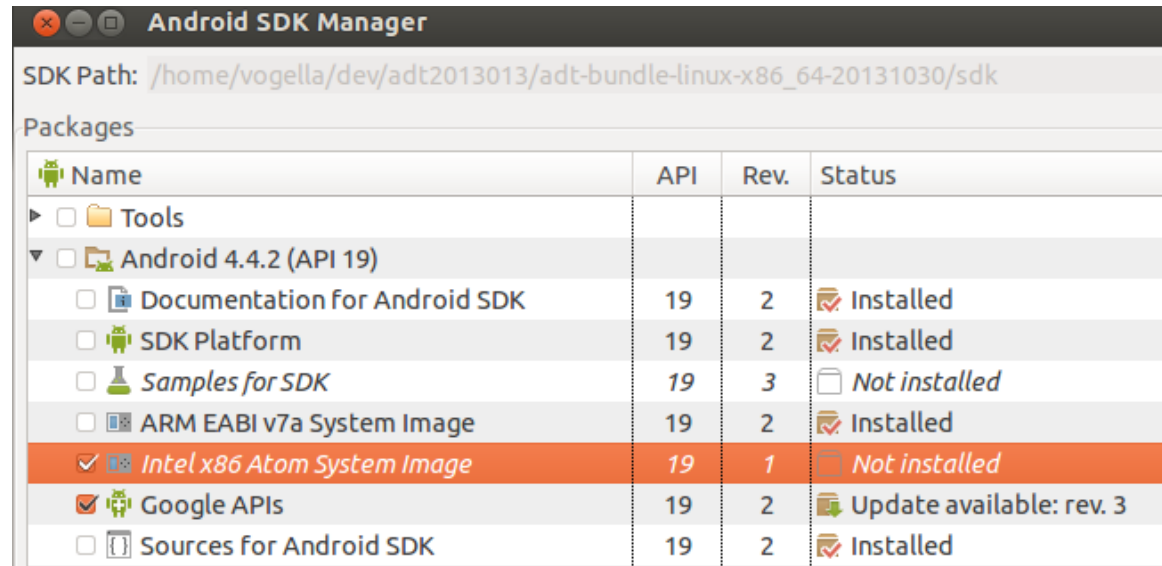
☐ Override the existing AVD with the same name

## 5.5. Intel system image

It is possible to run an AVD with an image based on the ARM CPU architecture or based on the Intel CPI architecture.

An Android virtual device which uses the Intel system image is much faster in execution on Intel / AMD hardware compared to the ARM based system image. This is because the emulator does not need to translate the ARM CPU instructions to the Intel / AMD CPU on your computer.

This image can be installed via the Android SDK Manager as depicted in the following screenshot.



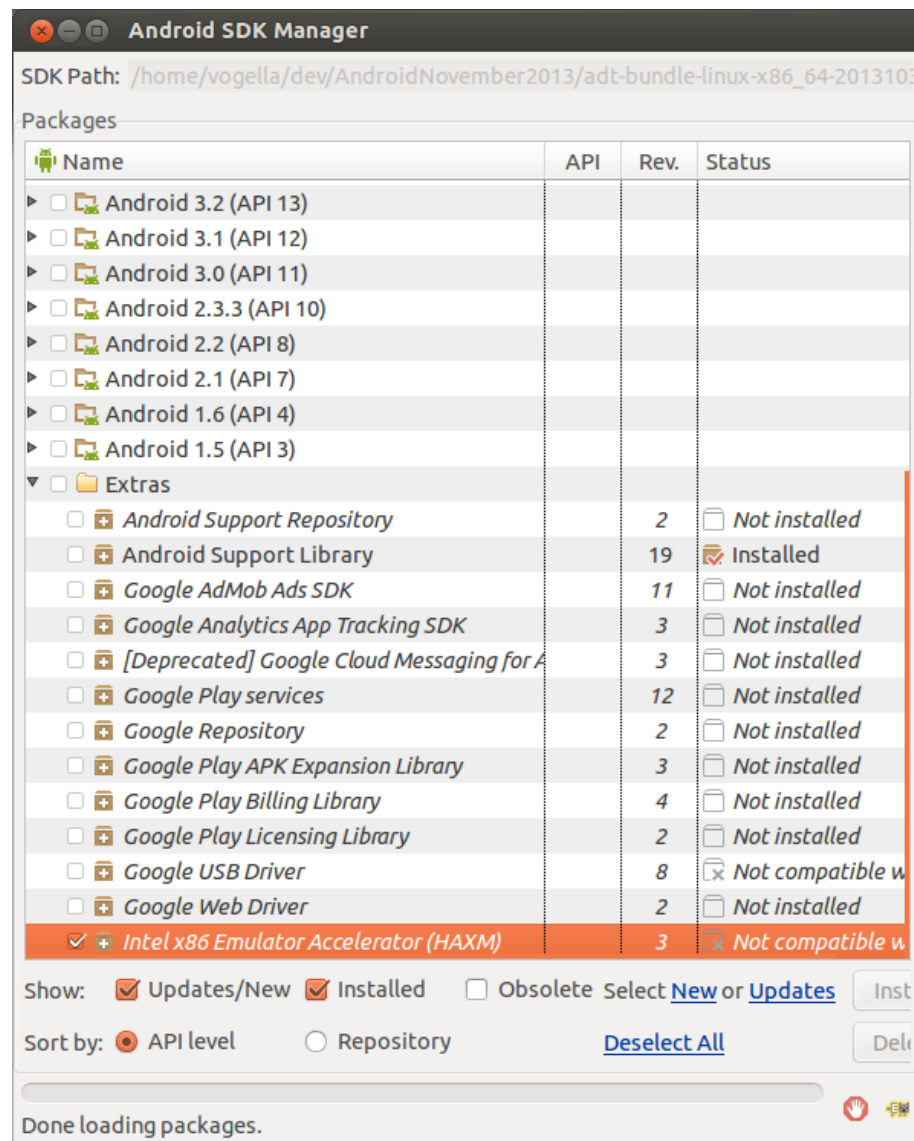
### Note

An Intel image is not available for all API levels.



### Note

At the time of this writing you also need to download and install extra drivers for MS windows.



After the download you find the driver in your Android installation directory in the *extras/intel* folder. You need to install the drivers by running starting the .exe file.



## Warning

This additional installation step is required on Window to accelerate the Intel emulator.

Only downloading the driver via the Android does not make a difference.

After the download you can create a new AVD based on the intel emulator. The emulator does not start faster but is way faster during the execution of your Android application.

Create new Android Virtual Device (AVD)

AVD Name:

Intent

Device:

Nexus 4 (4.7", 768 × 1280: xhdpi)

Target:

Android 4.4.2 - API Level 19

CPU/ABI:

Intel Atom (x86)

Keyboard:

☒ Hardware keyboard present

Skin:

☒ Display a skin with hardware controls

Front Camera:

None

Back Camera:

None

Memory Options:

RAM: 1907VM Heap: 64

Internal Storage:

200MiB

SD Card:

☒ Size: 200MiB

☐ File: Browse...

Emulation Options:

☐ Snapshot☐ Use Host GPU

☐ Override the existing AVD with the same name

Cancel

OK



### Tip

After the download you may need to restart your development environment to be able



to create an AVD with the intel emulator.



### Tip

Linux requires a more complex setup. For a detailed installation description see the [Intel emulator installation guide](#) which also includes detailed instructions for Windows.

## 5.6. Alternative emulator

There are alternatives to the default Android emulator available. For example, the [Genymotion emulator](#) is relatively fast in startup and execution of Android projects.

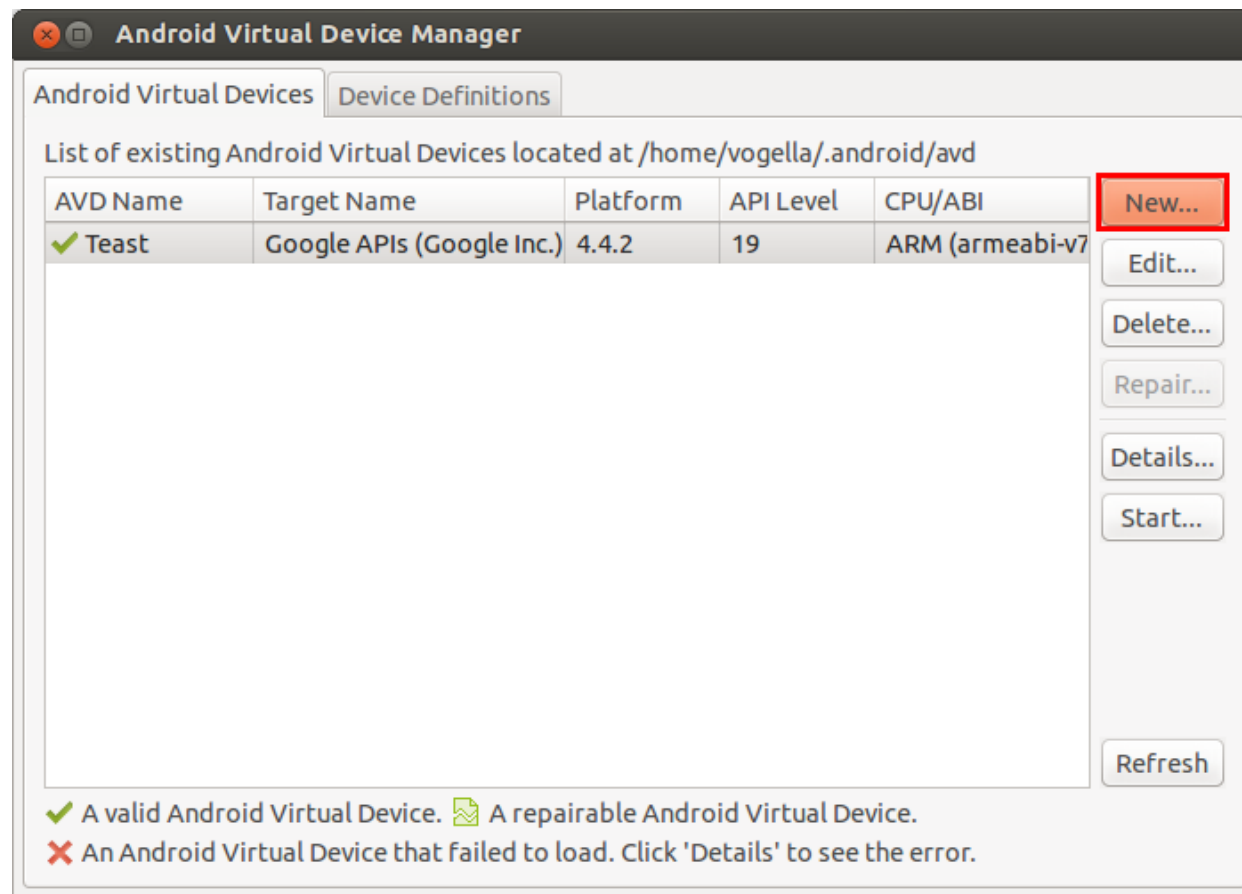
# 6. Exercise: Create an Android Virtual Device (AVD)

## 6.1. Target

In this exercise you create and start an AVD. Even if you have a real Android device available, you should get familiar with the creation and usage of AVDs. Virtual devices give you the possibility to test your application for selected Android versions and a specific configurations.

## 6.2. Create an AVD

Define a new Android Virtual Device (AVD) by opening the *AVD Manager* via *Window* → *Android Virtual Device Manager* and by pressing the *New* button.



Enter values similar to the following screenshot.

**Create new Android Virtual Device (AVD)**

AVD Name:

Device:

Target:

CPU/ABI:

Keyboard: ☒ Hardware keyboard present

Skin: ☒ Display a skin with hardware controls

Front Camera:

Back Camera:

Memory Options: RAM:  VM Heap:

Internal Storage:

SD Card:

☒ Size:

☐ File:

Emulation Options: ☐ Snapshot ☒ **Use Host GPU**

☐ Override the existing AVD with the same name



### Tip

Ensure that the *Use Host GPU* option is selected. This allows the AVD to use the



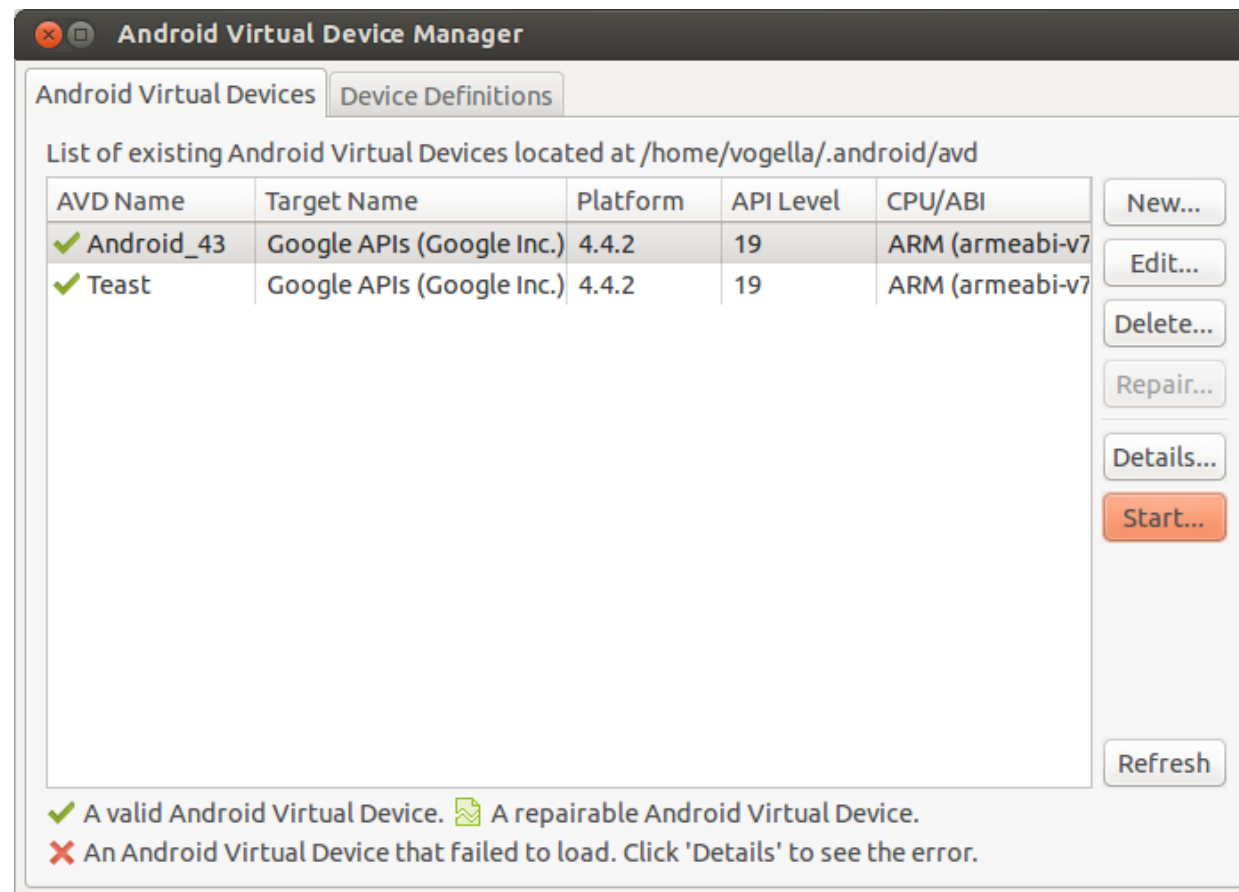


graphical processing unit (GPU) of your computer and makes rendering much faster.

Afterwards press the *OK* button. This will create the AVD configuration and display it under the list of available virtual devices.

## 6.3. Start your AVD

Select your new entry and press the *Start...* button. Select *Launch* in the following dialog.



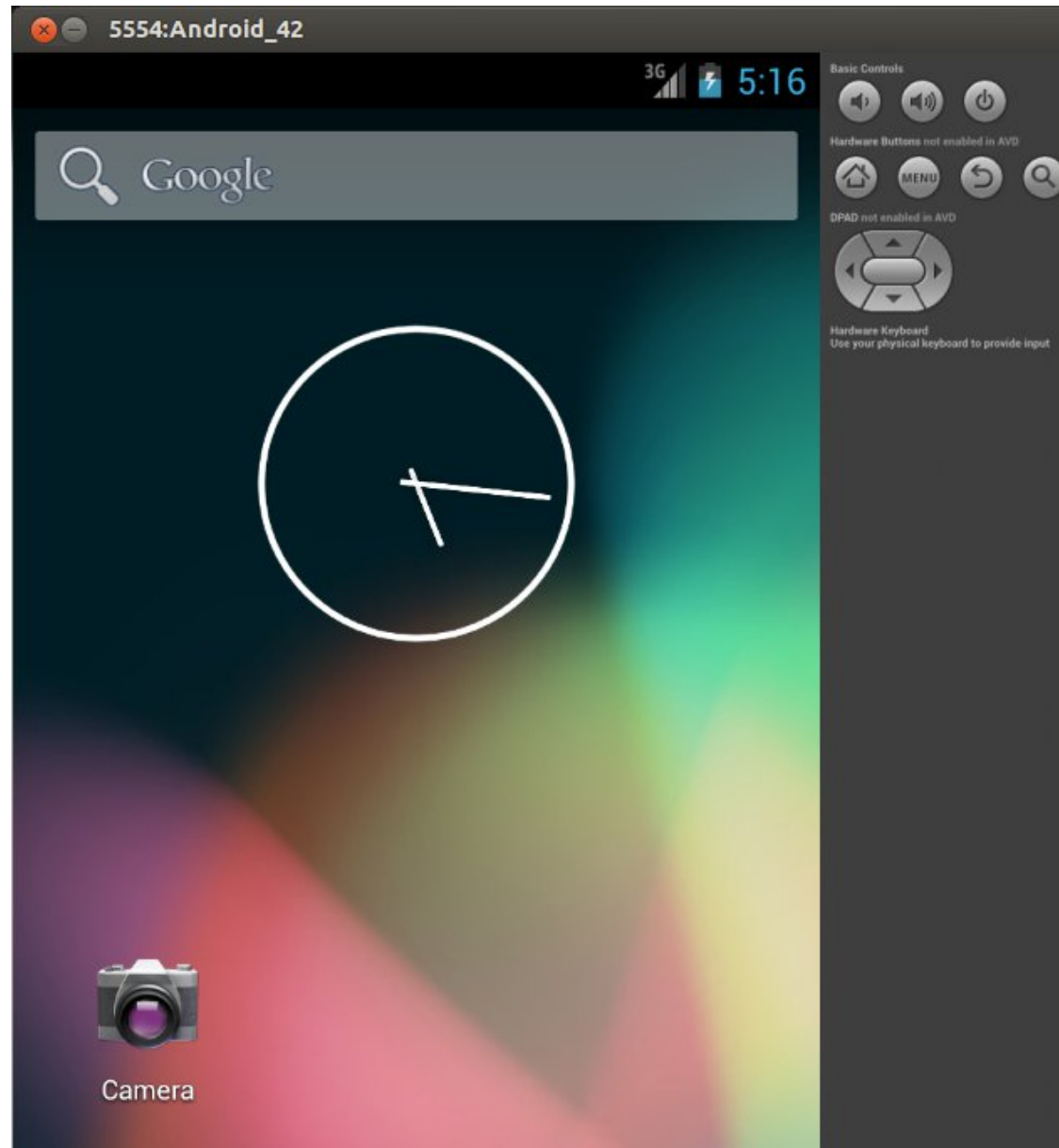
### Warning

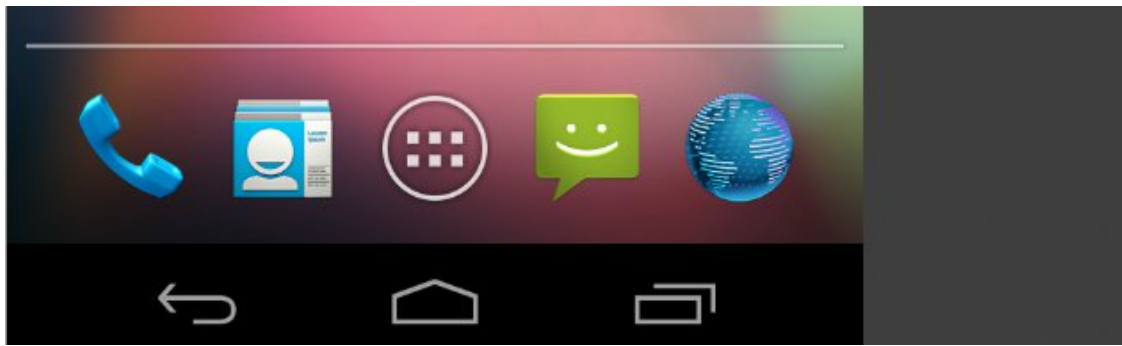
Do not interrupt this startup process, as this might corrupt the AVD. The first start may

Are you a developer? Try out the [HTML to PDF API](#)

take up to 10 minutes on an older machine. On a modern machine it typically takes 1-3 minutes for a new AVD to start.

After the AVD has started, you can control the GUI with the mouse. The emulator also provides access to the phone buttons via a menu on the right side of the emulator.





### Tip

Once started, don't stop the AVD during your development. If you change your application and want to test a new version, you simply re-deploy your application on the AVD.

## 7. Exercise: Create an Android application in Eclipse

### 7.1. Using the Android project wizard

The Android tooling in Eclipse provides wizards for Android applications. In this exercise you use the project creation wizard to create an Android application based on a template.



### Note

The purpose of this exercise is to demonstrate the development process. The created artifacts are explained later in this book.

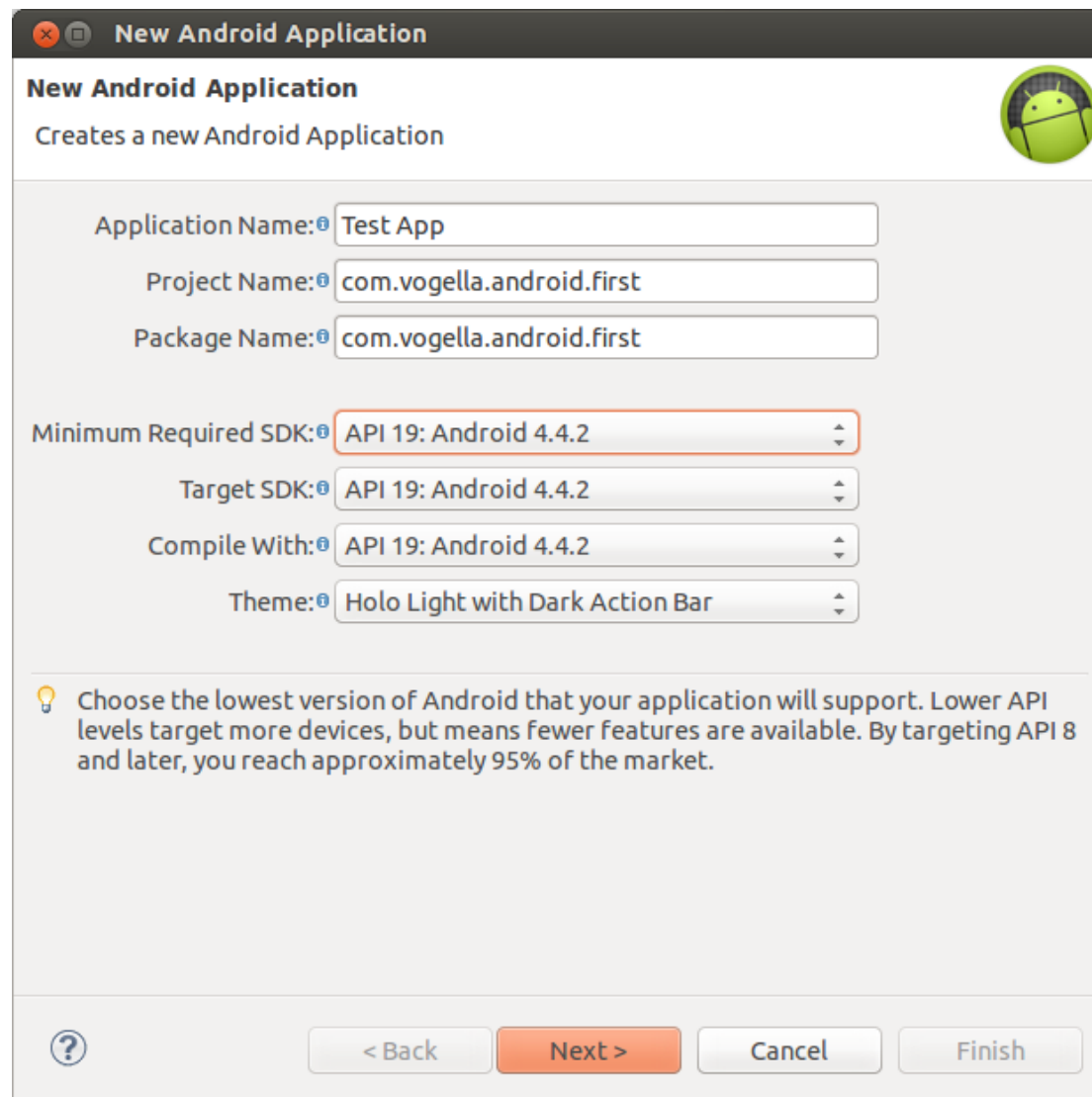
### 7.2. Create an Android project

To create a new Android project select *File* → *New* → *Other...* → *Android* → *Android Application Project* from the

menu. Enter the fitting data from the following table in the first wizard page.

**Table 2. Setting for your Android project**

Property	Value
Application Name	Test App
Project Name	com.vogella.android.first
Package Name	com.vogella.android.first
API (Minimum, Target, Compile with)	Latest



**New Android Application**  
Creates a new Android Application

Application Name:

Project Name:


Package Name:


Minimum Required SDK:

Target SDK:

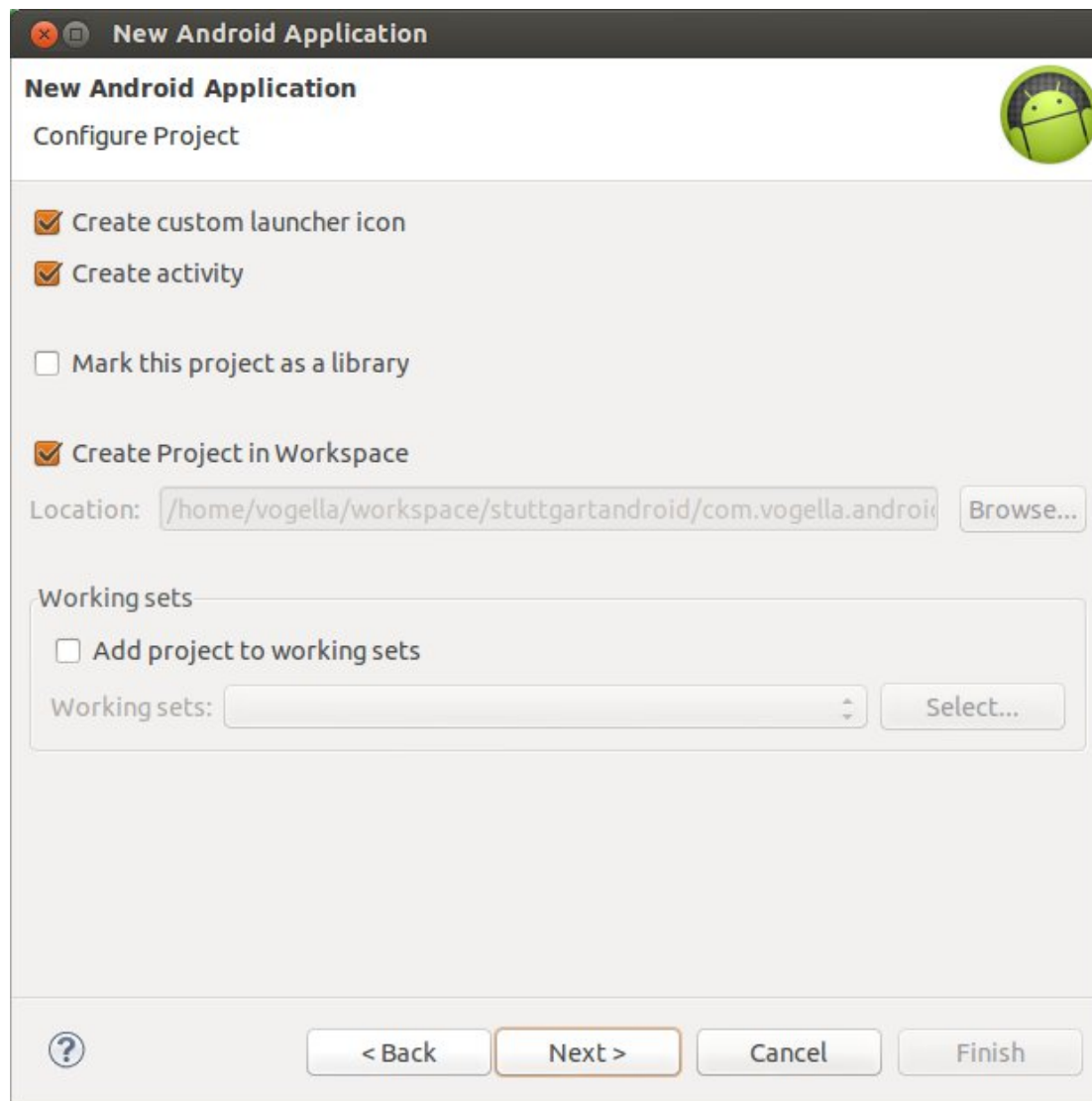
Compile With:

Theme:

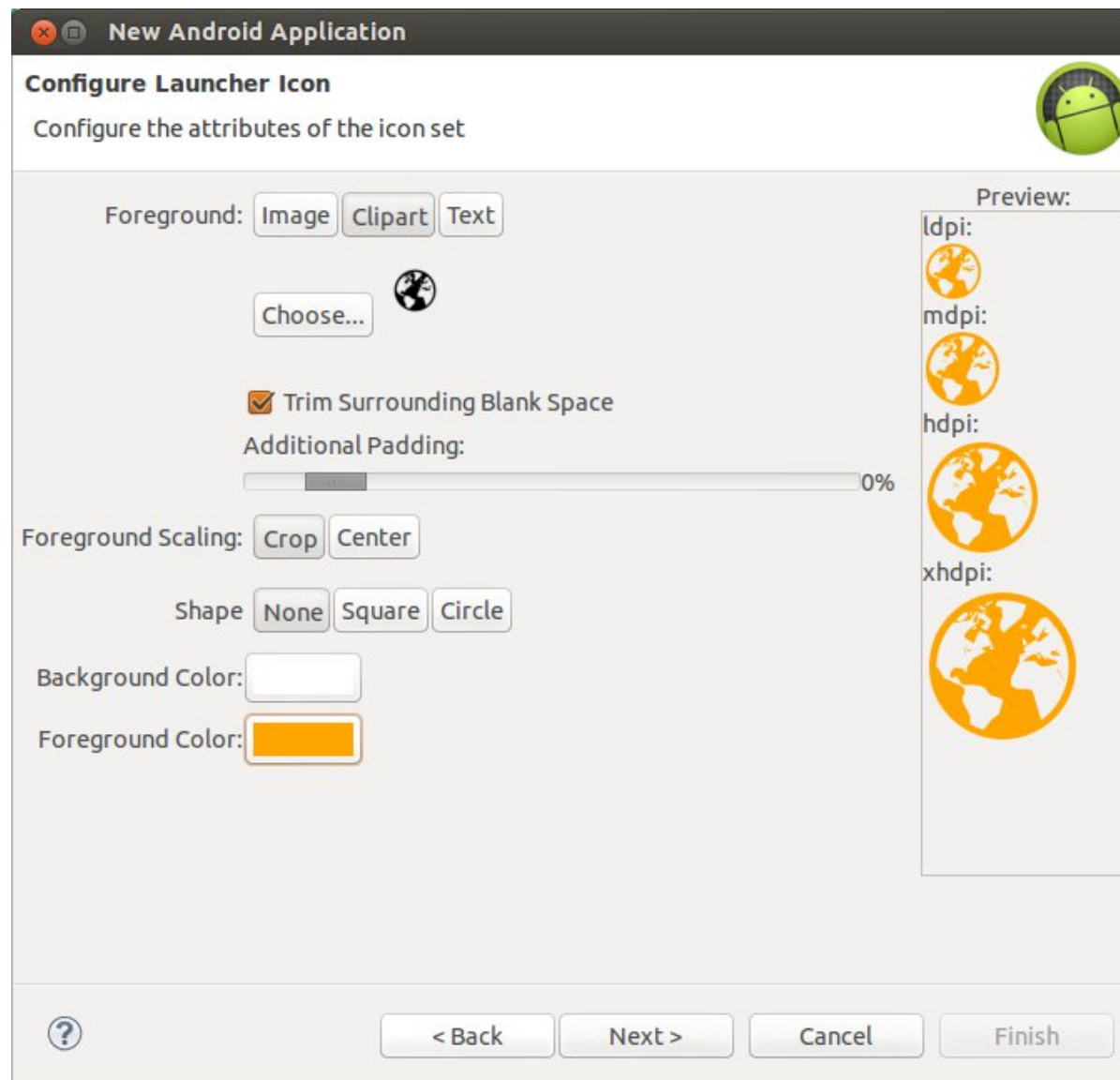
 Choose the lowest version of Android that your application will support. Lower API levels target more devices, but means fewer features are available. By targeting API 8 and later, you reach approximately 95% of the market.



Press the *Next* button and ensure that you have enabled the *Create custom launcher icon* and *Create activity* checkboxes.



On the wizard page for the launcher icon, create an application icon of your choice. The following screenshot shows an example for a possible result.



Press the *Next* button and select the *Empty Activity* template. Press the *Next* button to proceed.

New Android Application

Create Activity

Select whether to create an activity, and if so, what kind of activity.

☒ Create Activity

Blank Activity

Empty Activity

Fullscreen Activity

Master/Detail Flow

Empty Activity

Creates a new empty activity

?

< Back

Next >

Cancel

Finish

Enter the following data in the dialog for the template. The selection is depicted in the screenshot after the table.

**Table 3. Values for the template**

Parameter	Value
Activity	MainActivity

open in browser PRO version Are you a developer? Try out the [HTML to PDF API](#)


pdfcrowd.com



New Android Application

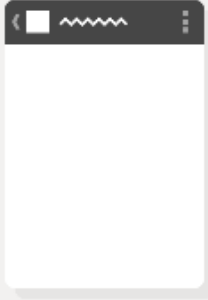
Empty Activity


Creates a new empty activity




Activity Name@ MainActivity

Layout Name@ activity\_main



 The name of the activity class to create



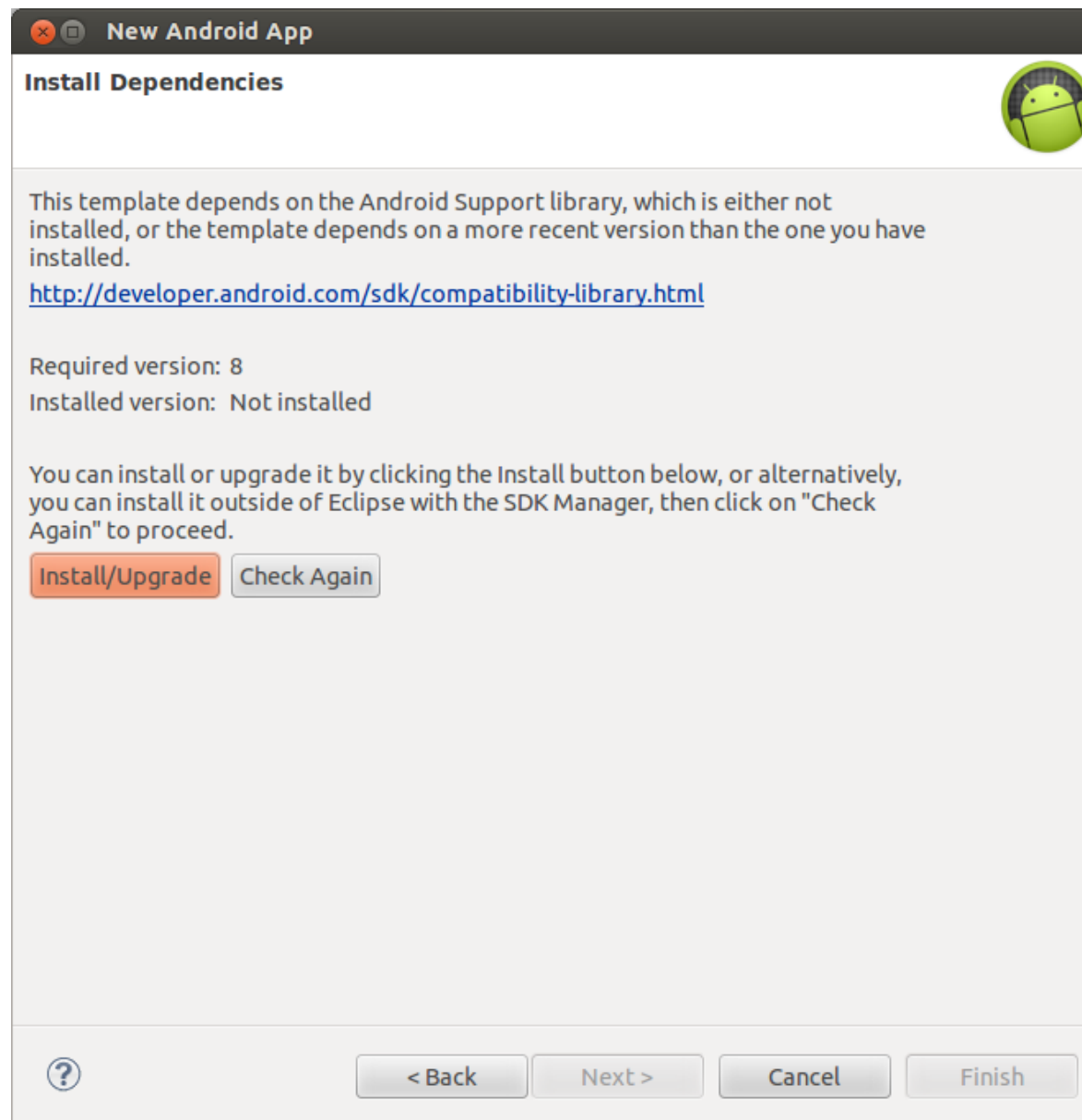
< Back

Next >

Cancel

Finish

Press the *Finish* button. The wizard may prompt you to install the support library. If so, select to install it.

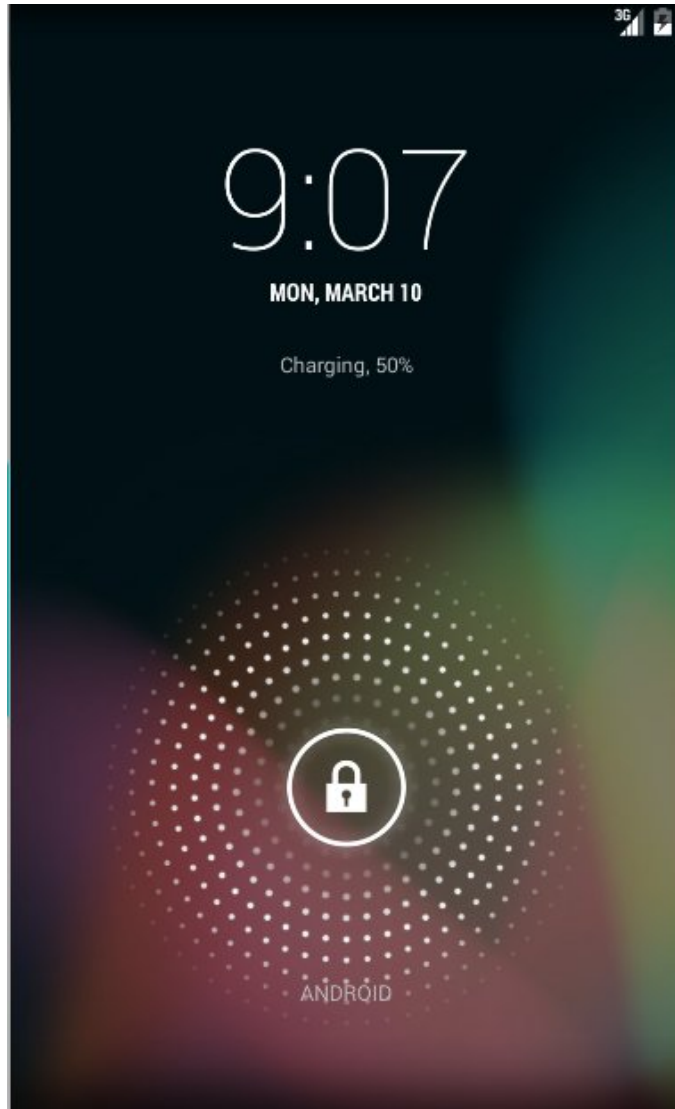


## 8. Exercise: Start the generated Android application

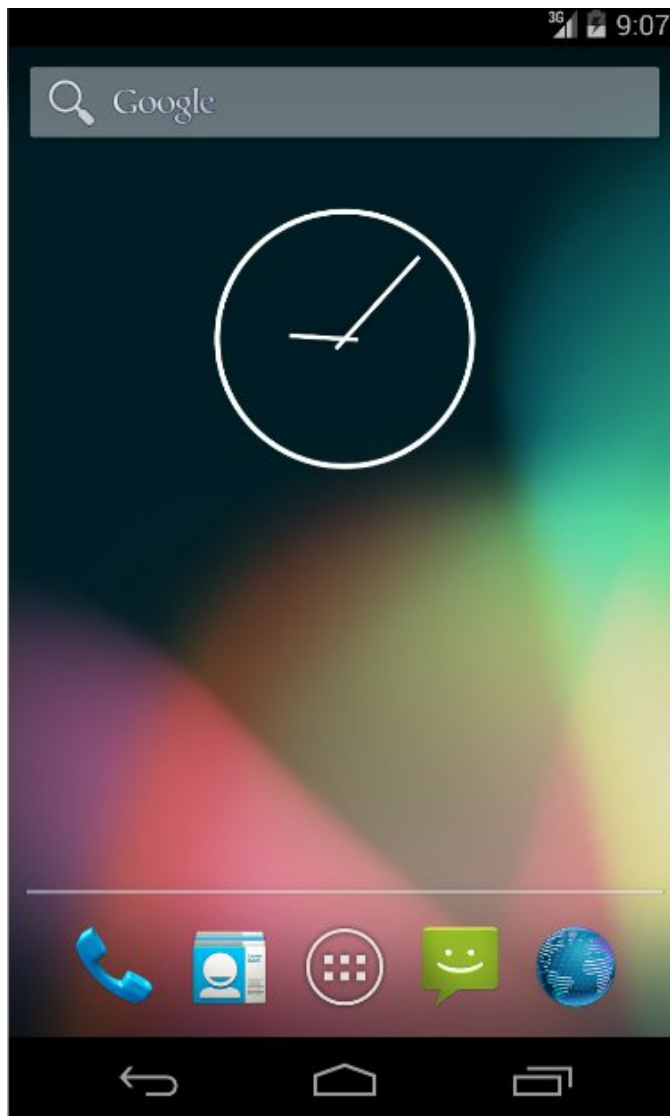
### 8.1. Start the AVD

If you have not yet done so far, create and start an Android virtual device (AVD). The Android version you select should fit to the minimum API version of your Android application.

After startup you see the welcome screen of your AVD as depicted in the following screenshot.

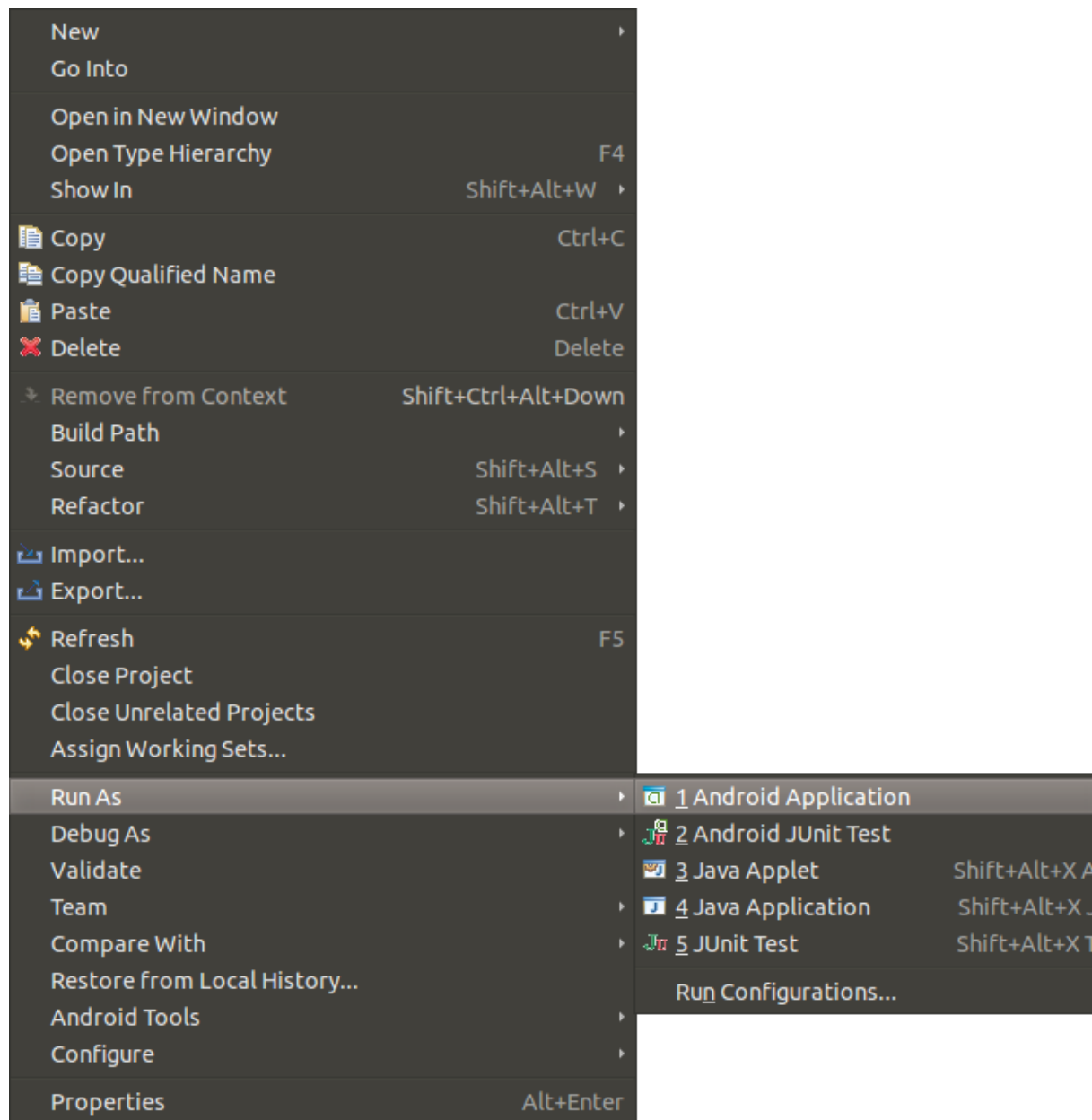


Once you AVD is ready, unlock your emulator.

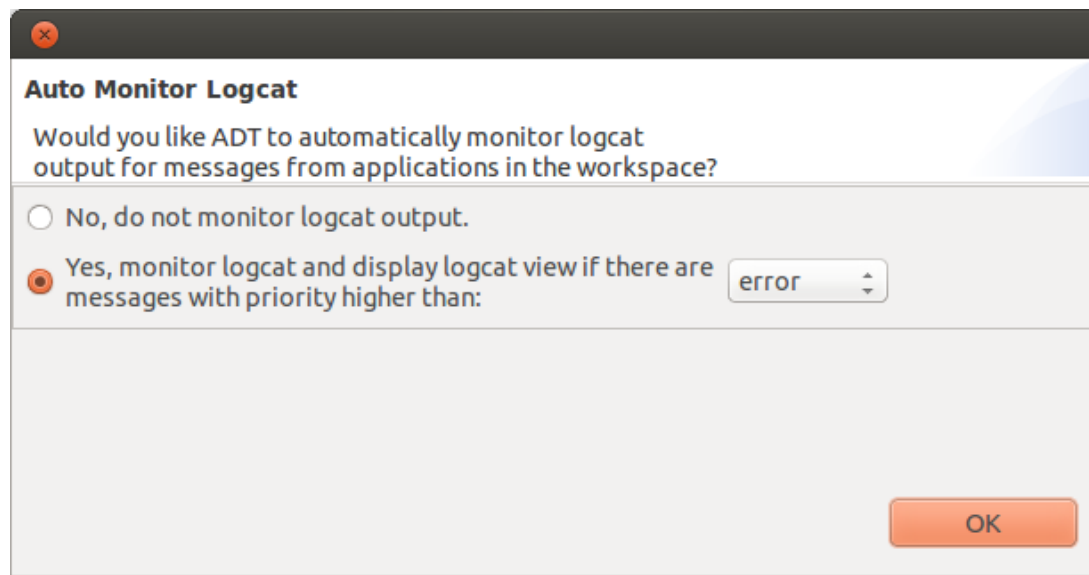


## 8.2. Start the application

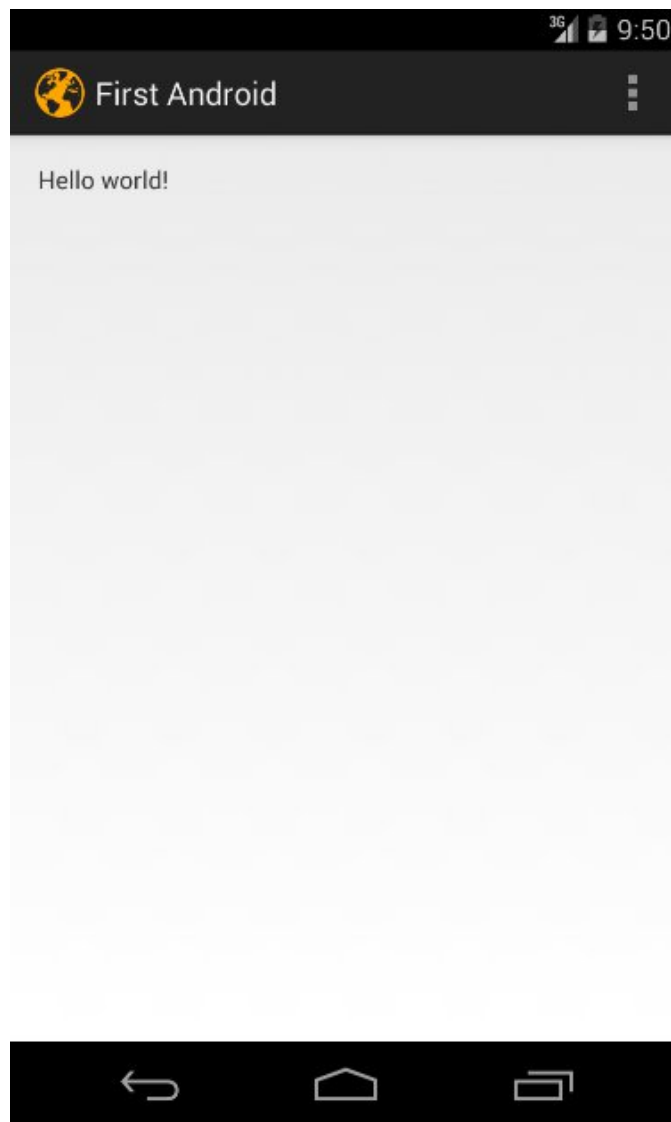
Select your Android project, right click on it, and select *Run-As* → *Android Application*.



You may be prompted whether the Android Developer Tools should monitor messages. Select Yes in this case and press the OK button.



This starts your application on the AVD. The started application is a very simple application which just shows the *Hello, world!* String.



## 9. Solving Android development problems

Things are not always working as they should. You find a list of typical Android development problems and their solution under the following link: [Solutions for common Android development problems](#).

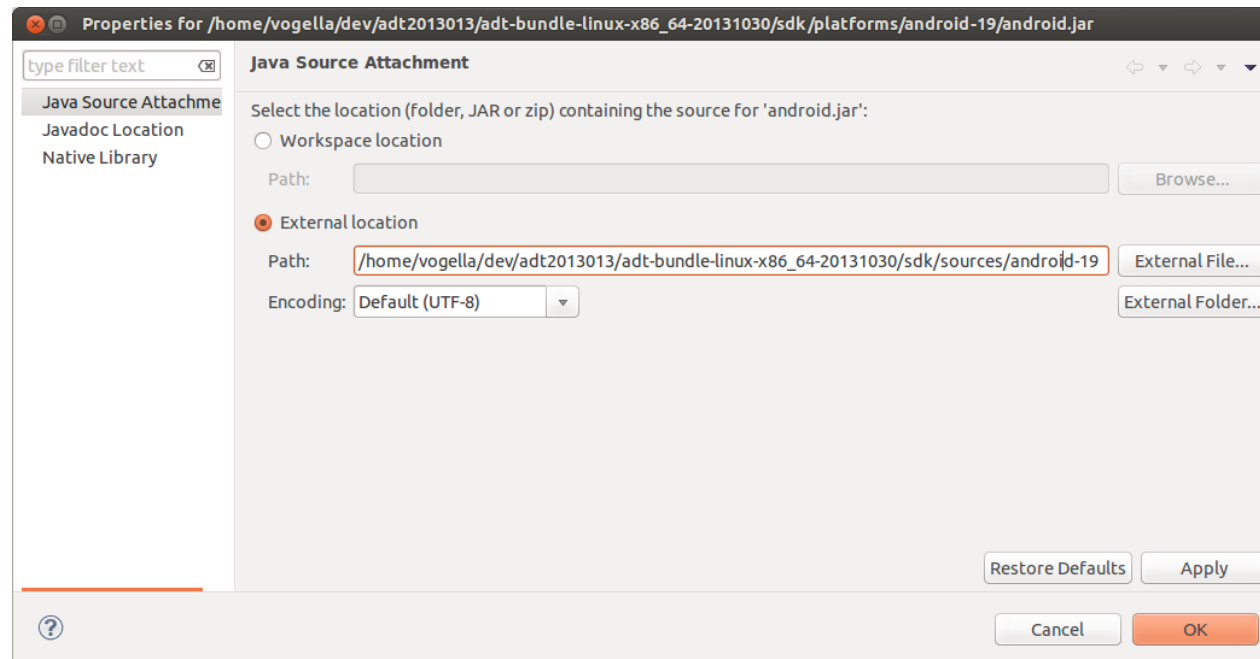
## 10. Connect with the Android source code

## 10.1. Connect the source to your Android JAR file

Now that you have already created an Android Application Project, you are able to connect it with the Android source code.

To connect the sources with the *android.jar* file in your Android project, right-click on your *android.jar* in the *Package Explorer* view and select *Properties* → *Java Source Attachment*.

Select *External location* and press the *External Folder...* button. Browse to the *path\_to\_android\_sdk/sources/android-xx* location and press the *OK* button.



## 10.2. Validate

Validate that you can see the Android source code. For example, open the *view* class via the *Open Type* dialog (Shortcut: **Ctrl+Shift+T**) and ensure that you can see the source code.

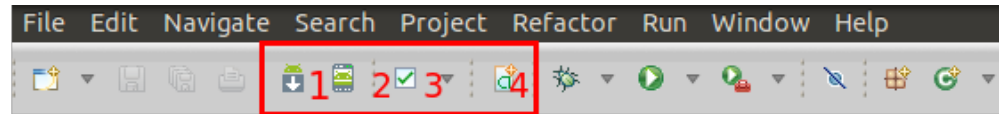
# 11. Integration of ADT into Eclipse

## 11.1. Android integration into the Java perspective



The Android tooling integrates itself into the Java perspective. It adds toolbar buttons to manage your Android configuration with the *Android SDK manager*, create new AVDs and configuration files as well as contains wizards for creating new Android projects and artifacts.

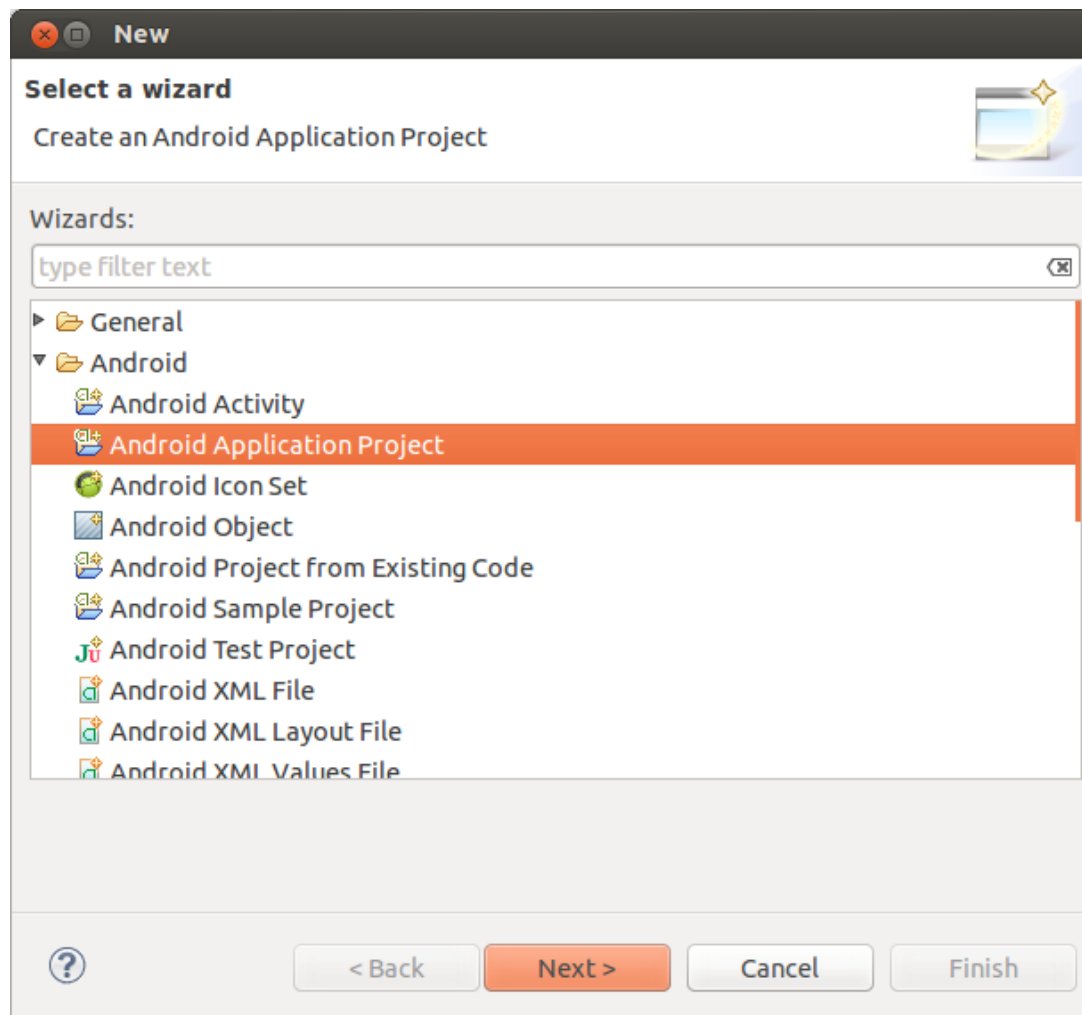
The following screenshot highlights the additional Android toolbars with a small description of these buttons.



- 1.) Android SDK manager
- 2.) Manage and start AVDs
- 3.) Check for your project for errors (Lint)
- 4.) New Android resource file

## 11.2. Android wizards

You find the Android specific wizards under *File* → *New* → *Other...* → *Android* as depicted in the following screenshot. These wizards allow you to create Android projects and other Android artifacts.

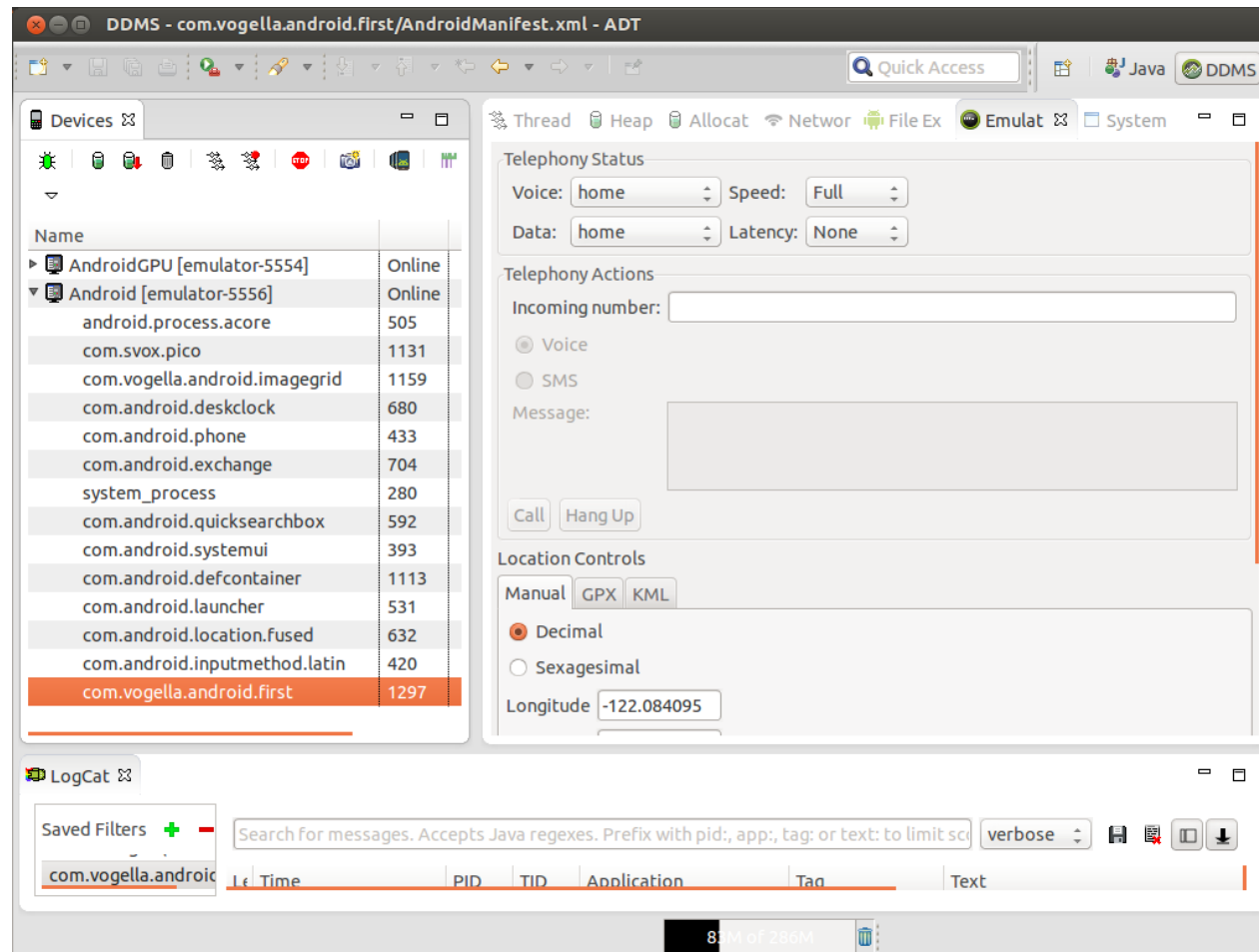


## 12. DDMS perspective

### 12.1. Android perspective

ADT adds the DDMS (Dalvik Device Monitoring Service) perspective to Eclipse for interacting with your Android (virtual) device and your Android application program. Select *Window* → *Open Perspective* → *Other...* → *DDMS* to open this perspective. It groups several Eclipse views which can also be used independently.

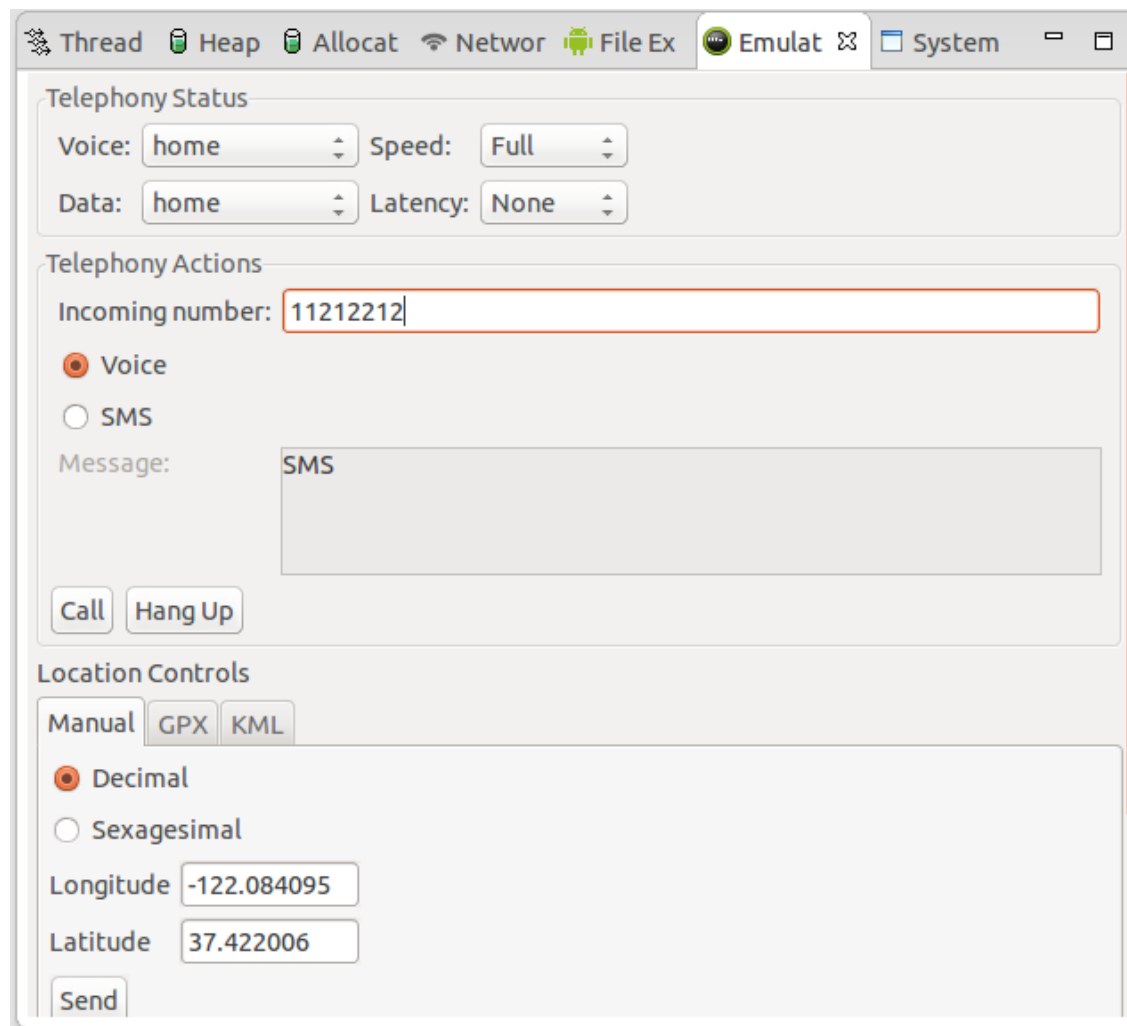
On the left side it shows you the connected Android devices and the running processes on the device. The right side is a stack of views with different purposes. You can select processes and trigger actions from the toolbar, e.g., start a trace or stop the process.



The following description highlights a few views in this perspective. Others are described once they are needed.

## 12.2. Emulator Control

The *Emulator Control* view allows you to simulate phone calls and SMS on the AVD. It also has the possibility to simulate the current geo position.



## 12.3. File explorer

The file explorer allows you to browse the file system on your Android virtual device.

Threads Heap Allocation Tra Network Stati File Explorer Emulat					
Name	Size	Date	Time	Permissions	Info
▶  acct		2013-06-12	21:48	drwxr-xr-x	
▶  cache		2013-06-12	21:48	drwxrwx---	
▶  config		2013-06-12	21:48	dr-xr-x---	
d		2013-06-12	21:48	lrwxrwxrwx	-> /sys/kernel/debug
▶  data		2013-06-12	21:49	drwxrwx-x	
default.prop	116	1970-01-01	00:00	-rw-r--r--	
▶  dev		2013-06-12	21:48	drwxr-xr-x	
etc		2013-06-12	21:48	lrwxrwxrwx	-> /system/etc
init	109412	1970-01-01	00:00	-rwxr-x---	
init.goldfish.rc	2487	1970-01-01	00:00	-rwxr-x---	

WE DON'T SHARE

YOUR FINANCIAL INFORMATION WITH STORES

Sign Up for Free

## 13. Parts of a Android application

### 13.1. Android application

An Android *application* is a single installable unit which can be started and used independently of other Android applications.

An Android application can have one application class which is instantiated as soon as the application starts and it is the last component which is stopped during application shutdown.

An Android application consists of Android software components and resource files.

Android application components can connect to components of other Android applications based on a task description (Intent). This way they can create cross-application tasks. The integration of these components can be done in a way that the Android application can still work flawless, even if the additional components are not installed or if different components perform the same task.

## 13.2. Android software components

The following software components can be defined in Android applications.

- Activities
- Services
- Broadcast receivers (short: receivers)
- Content providers (short: providers)

## 13.3. Context

Instances of the class `android.content.Context` provide the connection to the Android system which executes the application. It also gives access to the resources of the project and the global information about the application environment.

For example, you can check the size of the current device display via the `Context`.

The `Context` class also provides access to Android *services*, e.g., the alarm manager to trigger time based events.

Activities and services extend the `Context` class. Therefore they can be directly used to access the `Context`.

# 14. Android application components overview

## 14.1. Activity

An *activity* is the visual representation of an Android application. An Android application can have several activities.

Activities use *views* and *fragments* to create the user interface and to interact with the user. Both elements are described in the next sections.

## 14.2. BroadcastReceiver

A *broadcast receiver* (receiver) can be registered to listen to system messages and intents. A receiver gets notified by the Android system if the specified event occurs.

For example, you can register a receiver for the event that the Android system finished the boot process. Or you

can register for the event that the state of the phone changes, e.g., someone is calling.

### 14.3. Service

A *service* performs tasks without providing an user interface. They can communicate with other Android components, for example, via broadcast receivers and notify the user via the notification framework in Android.

### 14.4. ContentProvider

A *content provider* (provider) defines a structured interface to application data. A provider can be used for accessing data within one application, but can also be used to share data with other applications.

Android contains an SQLite database which is frequently used in conjunction with a content provider. The SQLite database would store the data, which would be accessed via the provider.

## 15. Base user interface components in Android

The following description gives an overview of the most important user interface related component and parts of an Android application.

### 15.1. Activity

Activities are the base for the user interface in Android. They have already been introduced in [Section 14.1, “Activity”](#).

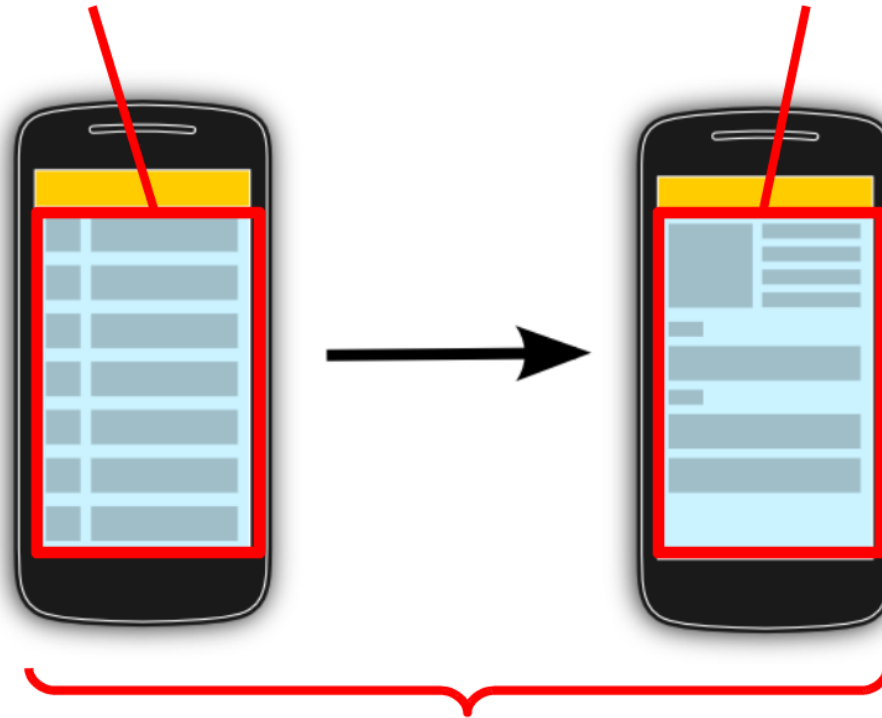
### 15.2. Fragments

Fragments are components which run in the context of an activity. A fragment encapsulates application code so that it is easier to reuse them and to support devices of different size.

The following picture shows an activity called *MainActivity*. On a smaller screen it shows only one fragment and allows the user to navigate to another fragment. On a wide screen it shows those two fragments immediately.

ListFragment

DetailsFragment

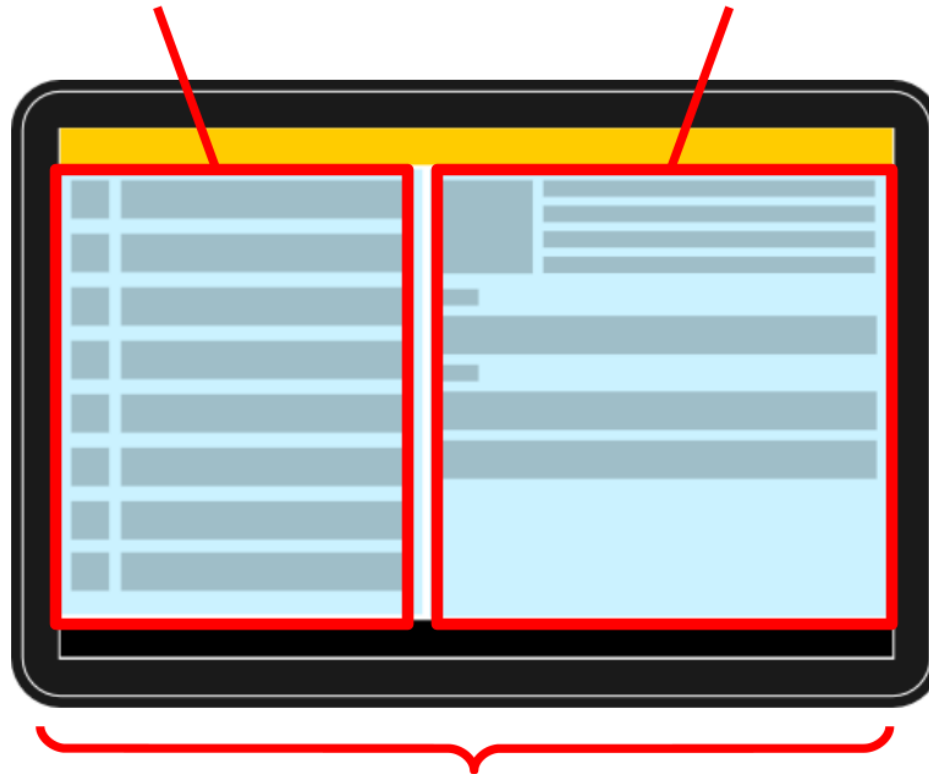


MainActivity



# ListFragment

# DetailsFragment



## MainActivity

### 15.3. Views and layout manager

*Views* are user interface widgets, e.g., buttons or text fields. Views have attributes which can be used to configure their appearance and behavior.

A *ViewGroup* is responsible for arranging other views. It is also known as *layout manager*. The base class for these layout managers is the `android.view.ViewGroup` class which extends the `android.view.View` class which is the base class for views.

Layout managers can be nested to create complex layouts.

### 15.4. Device configuration specific layouts

The user interface for activities is typically defined via XML files (layout files). It is possible to define layout files for different device configuration, e.g., based on the available width of the actual device running the application.

## 16. Other important Android elements

### 16.1. Home screen and lock screen widgets

*Widgets* are interactive components which are primarily used on the Android home screen. They typically display some kind of data and allow the user to perform actions with them. For example, a widget can display a short summary of new emails and if the user selects an email, it could start the email application with the selected email.

To avoid confusion with views (which are also called widgets), this text uses the term *home screen widgets*, if it speaks about widgets.

### 16.2. Live Wallpapers

*Live wallpapers* allow you to create animated backgrounds for the Android home screen.

## 17. The Android manifest

### 17.1. Configuration of your Android application

The components and settings of an Android application are described in the *AndroidManifest.xml* file. This file is known as the *manifest file* or the *manifest*.

The manifest also specifies additional metadata for the application, e.g., icons and the version number of the application.

This file is read by the Android system during installation of the application. The Android system evaluates this configuration file and determines the capabilities of the application.

### 17.2. Declaring components in the manifest file

All activities, services and content provider components of the application must be statically declared in this file. Broadcast receiver can be defined statically in the manifest file or dynamically at runtime in the application.

### 17.3. Permissions

The Android manifest file must also contain the required permissions for the application. For example, if the application requires network access, it must be specified here.

## 17.4. AndroidManifest.xml example file

The following listing shows an example for a simple *AndroidManifest.xml* file.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.rssfeed"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="19" />

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

    <application
        android:name="RssApplication"
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="RssfeedActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DetailActivity"
            android:label="Details" >
        </activity>
        <activity android:name="MyPreferenceActivity" >
        </activity>

        <service android:name="RssDownloadService" >
        </service>
    </application>

</manifest>
```

## 18. The Android manifest

## 18.1. Version and package

The *package* attribute defines the base package for the Java objects referred to in this file. If a Java object lies within a different package, it must be declared with the full qualified package name.

Google Play requires that every Android application uses its own unique package name. Therefore it is a good habit to use your reverse domain name here. This will avoid collisions with other Android applications.

*android:versionName* and *android:versionCode* specify the version of your application. *versionName* is what the user sees and can be any string.

*versionCode* must be an integer. The Android Market determines whether it should perform an update of the applications for the existing installation based on that *versionCode*. You typically start with "1" and increase this value by one if you roll-out a new version of your application.

## 18.2. Application and components

The *<application>* section allows to define metadata for your application and optionally define an explicit application class. It is also a container for declaring your Android components.

The *<activity>* tag defines an activity component. The *name* attribute points to class, which (if not fully qualified) is relative to the package defined in the *package* attribute.

The *intent filter* part in the Android manifest file, tells the Android runtime that this activity should be registered as a possible entry point into the application and made available in the launcher of the Android system. The action defines that it (*android:name="android.intent.action.MAIN"* ) can be started and the *category* *android:name="android.intent.category.LAUNCHER"* parameter tells the Android system to add the activity to the launcher.

The *@string/app\_name* value refers to resource files which contain the actual value of the application name. The usage of a resource file makes it easy to provide different resources (e.g., strings, colors, icons) for different devices and makes it easy to translate applications.

Similar to the *<activity>* tag, you can use the service, receiver and provider to declare other Android components.

## 18.3. Minimum and target SDK

The *uses-sdk* section in the manifest allows you to specify the *minSdkVersion* and *targetSdkVersion* version of your application.

### Table 4. Minimum and target version

Value	Description
minSdkVersion	Define the minimum version of Android your application works on. This attribute is used as a filter in Google Play, i.e., a user cannot install your application on a device with a lower API level than specified in this attribute.
targetSdkVersion	Specifies the version on which you tested and developed. If it is not equal to the API version of the Android device, the Android system might apply forward- or backward-compatibility changes. It is good practice to always set this to the latest Android API version to take advantages of changes in the latest Android improvements.

## 18.4. Permissions

Your application can declare permissions with the `<permission>` tag and declare that it required a permission with the `<uses-permission>` tag.

## 18.5. Required device configuration

The `uses-configuration` section in the manifest allows you to specify required input methods for your device. For example, the following snippet would require that the device has a hardware keyboard.

```
<uses-configuration android:reqHardKeyboard="true"/>
```

The `uses-feature` section allows you to specify the required hardware configuration for your device. For example, the following snippet would require that the device has a camera.

```
<uses-feature android:name="android.hardware.camera" />
```

## 18.6. Installation location

Via the `installLocation` attribute of your application you can specify if your application can be installed on the external storage of the device. Use `auto` or `preferExternal` to permit this.



### Warning

In reality this option is rarely used, as an application installed on the external storage is

stopped once the device is connected to a computer and mounted as USB storage.

## 18.7. More info

You find more information about the attributes and sections of the manifest in the [Android Manifest documentation](#).

# 19. Resources

## 19.1. Resource files

Resources, like images and XML configuration files, are kept separate from the source code in Android applications.

Resource files must be placed in the `/res` directory in a predefined sub-folder. The specific sub-folder depends on type of resource which is stored.

The following table gives an overview of the supported resources and their standard folder prefixes.

**Table 5. Resources**

Resource	Folder	Description
Drawables	<code>/res/drawables</code>	Images (e.g., png or jpeg files) or XML files which describe a Drawable object.
Simple Values	<code>/res/values</code>	Used to define strings, colors, dimensions, styles and static arrays of strings or integers via XML files. By convention each type is stored in a separate file, e.g., strings are defined in the <code>res/values/strings.xml</code> file.
Layouts	<code>/res/layout</code>	XML files with layout descriptions are used to define the user interface for Activities and fragments.
Styles and Themes	<code>/res/values</code>	Files which define the appearance of your Android application.
Animations	<code>/res/anim</code>	Defines animations in XML for the animation API which allows to animate arbitrary properties of objects over time.
Raw data	<code>/res/raw</code>	Arbitrary files saved in their raw form. You access them via an <code>InputStream</code> object.

Menus	/res/menu	Defines the properties of entries for a menu.
-------	-----------	-----------------------------------------------

## 19.2. Resource example

The following listing is an example for file called *values.xml* in the */res/values* which defines a few String constants, a String array, a color and a dimension.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Test</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>

    <string-array name="operationsystems">
        <item>Ubuntu</item>
        <item>Android</item>
        <item>Microsoft Windows</item>
    </string-array>

    <color name="red">#ffff0000</color>

    <dimen name="mymargin">10dp</dimen>

</resources>
```

## 19.3. Resource qualifiers

You can also append additional qualifiers to the folder name to indicate that the related resources should be used for special configurations. For example, you can specify that layout file is only valid for a certain screen size.

## 19.4. Resource IDs and R.java

Every resource gets an ID assigned by the Android build system. The *gen* directory in an Android project contains the *R.java* references file which contains these generated values. These references are static integer values.

If you add a new resource file, the corresponding reference is automatically created in a *R.java* file. Manual changes in the *R.java* file are not necessary and will be overwritten by the tooling.

The Android system provides methods to access the corresponding resource files via these IDs.

For example, to access a String with the `R.string.yourString` ID in your source code, you would use the

`getString(R.string.yourString)` method defined on the `Context` class.

## 19.5. Good practices for resources IDs

The Android SDK uses the *camelCase* notation for most of its IDs, e.g., `buttonRefresh`. It is good practice to follow this approach.

## 19.6. System resources

Android also provides resources. These are called *system resources*. System resources are distinguished from local resources by the `android` namespace prefix. For example, `android.R.string.cancel` defines the platform string for a cancel operation.

# 20. Layout resource files

## 20.1. Activities and layouts

Android activities define their user interface with views (widgets) and fragments. This user interface can be defined via XML layout resource files in the `/res/layout` folder or via Java code. You can also mix both approaches.

Defining layouts via XML layout files is the preferred way. This separates the programming logic from the layout definition. It also allows the definition of different layouts for different devices.

## 20.2. XML layout files

A layout resource file is referred to as *layout*. A layout specifies the `ViewGroups`, `Views`, their relationship and their attributes via an XML representation.

The following code is an example for a simple layout file.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/mytext"
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

A layout is assigned to an activity via the `setContentView()` method calls, as demonstrated in the following example code.

```
package com.vogella.android.first;

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

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

}
```

## 20.3. Defining IDs

If a view needs to be accessed via Java code, you have to give the view a unique ID via the *android:id* attribute. To assign a new ID to a view use the *android:id* attribute of the corresponding element in the layout file. The following shows an example in which a button gets the *button1* ID assigned via the *android:id="@+id/button1"* parameter.

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Preferences" >
</Button>
```

By conversion this statement creates a new ID if necessary in the `R.java` file and assigns the defined ID to the corresponding view.

## 20.4. Good practice: Predefined IDs via a separate file

As described in the last section Android allows you to define IDs of user interface components dynamically in the layout files. To have one central place to define ID, you can also define them in a configuration file.

To control your IDs, you can also create a file, typically called *ids.xml*, in your */res/values* folder and define your IDs in this file. The following listing shows an example for such a file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <item name="button1" type="id"/>
</resources>
```

This allows you to use the ID in your layout file.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <Button
    android:id="@id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:layout_marginRight="27dp"
    android:text="Button" />

</RelativeLayout>
```

## 20.5. Performance considerations with layouts

Calculating the layout and drawing the views is a resource intensive operation. You should use the simplest layout possible to achieve good performance. For example, you should avoid nesting layout managers too deeply or avoid using complex layout managers in case a simple layout manager is sufficient.



## 21. Views

### 21.1. View class

A *view* in Android represents a widget, e.g., a button, or a layout manager.

All views in Android extend the `android.view.View` class. This class is relatively large (more than 18 000 lines of code) and provides a lot of base functionality for subclasses.

### 21.2. Standard views

The Android SDK provides standard views (widgets), e.g., via the `Button`, `TextView`, `EditText` classes.

It also includes complex widgets, for example, `ListView` or `GridView` to show structured data.

The main packages for views are part of the `android.view` namespace for all the base classes and `android.widget` for the default widgets of the Android platform.

### 21.3. Custom views

Developers can implement their own views by extending `android.view.View`.

## 22. Layout Manager and ViewGroups

### 22.1. What is a layout manager?

A layout manager is a subclass of `ViewGroup` and is responsible for the layout of itself and its child `Views`. Android supports different default layout managers.

## 22.2. Important layout managers

As of Android 4.0 the most relevant layout managers are `LinearLayout`, `FrameLayout`, `RelativeLayout` and `GridLayout`.

`AbsoluteLayout` is deprecated and `TableLayout` can be implemented more effectively via `GridLayout`.



### Warning

`RelativeLayout` is a complex layout manager and should only be used if such a complex layout is required, as it performs a resource intensive calculation to layout its children.

## 22.3. Layout attributes

All layouts allow the developer to define attributes. Children can also define attributes which may be evaluated by their parent layout.

Children can specify their desired width and height via the following attributes.

**Table 6. Width and height definition**

Attribute	Description
<code>android:layout_width</code>	Defines the width of the widget.
<code>android:layout_height</code>	Defines the height of the widget.

Widgets can use fixed sizes, e.g., with the *dp* definition, for example, *100dp*. While *dp* is a fixed size it will scale with different device configurations.

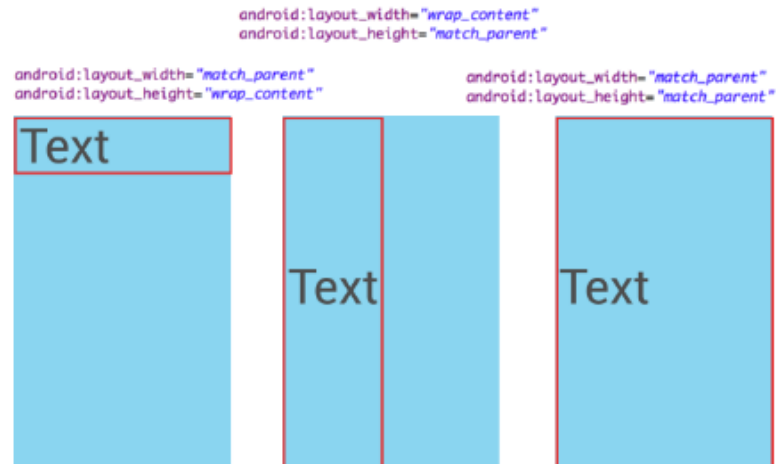
The *match\_parent* value tells the application to maximize the widget in its parent. The *wrap\_content* value tells the layout to allocate the minimum amount so that the widget is rendered correctly. The effect of these elements is demonstrated in the following graphics.

# wrap\_content

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
/>
```



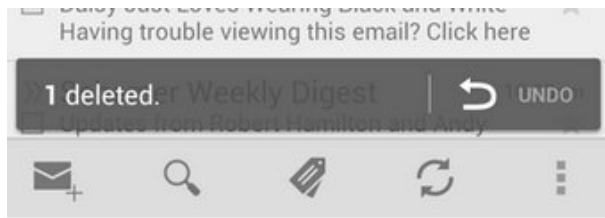
# match\_parent



## 22.4. FrameLayout

FrameLayout is a layout manager which draws all child elements on top of each other. This allows to create nice visual effects.

The following screenshot shows the Gmail application which uses FrameLayout to display several button on top of another layout.



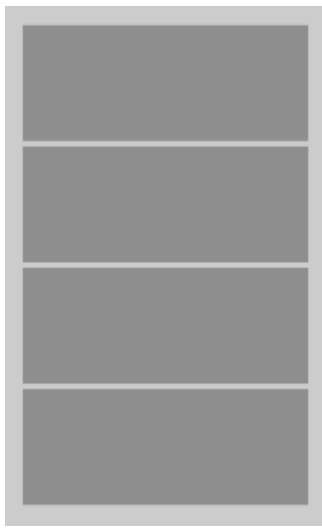
## 22.5. LinearLayout

`LinearLayout` puts all its child elements into a single column or row depending on the `android:orientation` attribute. Possible values for this attribute are *horizontal* and *vertical* while *horizontal* is the default value.

If horizontal is used, the child elements are layouted as indicated by the following picture.



Vertical would result in a layout as depicted in the following picture.



`LinearLayout` can be nested to achieve more complex layouts.

`LinearLayout` supports assigning a weight to individual children via the `android:layout_weight` layout parameter. This value specifies how much of the extra space in the layout is allocated to the `View`. If, for example, you have two widgets and the first one defines a `layout_weight` of 1 and the second of 2, the first will get 1/3 of the available space and the other one 2/3. You can also set the `layout_width` to zero to always have a certain ratio.

## 22.6. RelativeLayout

`RelativeLayout` allows to position the widget relative to each other. This can be used for complex layouts.

A simple usage for `RelativeLayout` is if you want to center a single component. Just add one component to the `RelativeLayout` and set the `android:layout_centerInParent` attribute to `true`.

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

    <ProgressBar
        android:id="@+id/progressBar1"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
    />
```

```
</RelativeLayout>
```

## 22.7. GridLayout

`GridLayout` was introduced with Android 4.0. This layout allows you to organize a view into a Grid. `GridLayout` separates its drawing area into: rows, columns, and cells.

You can specify how many columns you want to define for each view, in which row and column it should be placed as well as how many columns and rows it should use. If not specified, `GridLayout` uses defaults, e.g., one column, one row and the position of a view depends on the order of the declaration of the views.

The following layout file defines a layout using `GridLayout`.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/GridLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="4"
    android:useDefaultMargins="true" >

    <TextView
        android:layout_column="0"
        android:layout_columnSpan="3"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="40dp"
        android:layout_row="0"
        android:text="User Credentials"
        android:textSize="32dip" />

    <TextView
        android:layout_column="0"
        android:layout_gravity="right"
        android:layout_row="1"
        android:text="User Name: " >
</TextView>

    <EditText
        android:id="@+id/input1"
        android:layout_column="1"
        android:layout_columnSpan="2"
        android:layout_row="1"
        android:ems="10" />

    <TextView
        android:layout_column="0"
        android:layout_gravity="right"
        android:layout_row="2"
        android:text="Password: " >
```



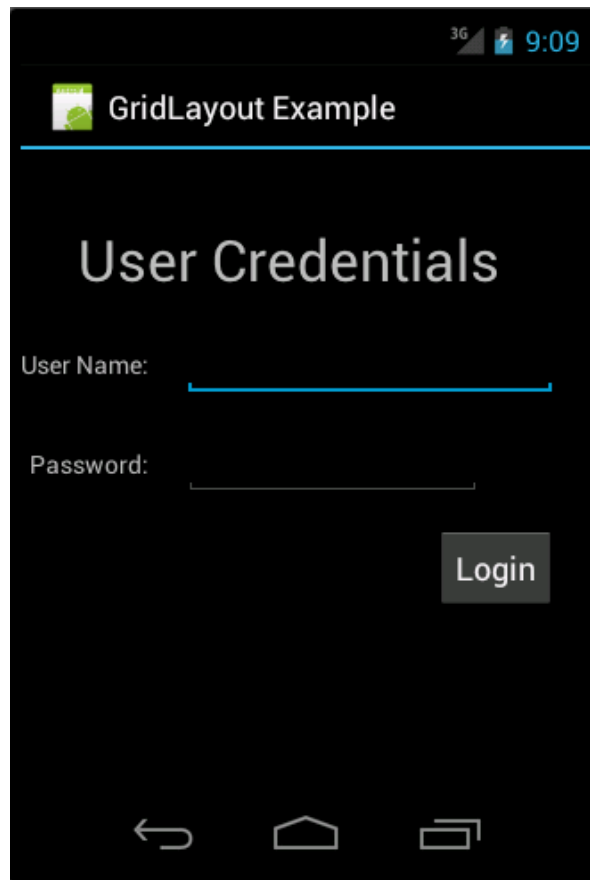
```
</TextView>

<EditText
    android:id="@+id/input2"
    android:layout_column="1"
    android:layout_columnSpan="2"
    android:layout_row="2"
    android:inputType="textPassword"
    android:ems="8" />

<Button
    android:id="@+id/button1"
    android:layout_column="2"
    android:layout_row="3"
    android:text="Login" />

</GridLayout>
```

This creates a user interface similar to the following screenshot.



## 22.8. ScrollView

The `ScrollView` class can be used to contain one `View` that might be too big to fit on one screen. In this case `ScrollView` will display a scroll bar to scroll the content.

Of course this `view` can be a layout which can then contain other elements.

The following code shows an example layout file which uses a `ScrollView`.

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

    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="8dip"
        android:paddingRight="8dip"
        android:paddingTop="8dip"
        android:text="This is a header"
        android:textAppearance="?android:attr/textAppearanceLarge" >
    </TextView>

</ScrollView>
```

The `android:fillViewport="true"` attribute ensures that the `scrollview` is set to the full screen even if the elements are smaller than one screen.

## 23. Exercise: Use layouts and add interaction

### 23.1. Review layout

In your `com.vogella.android.first` project, open the `activity_main.xml` layout file in the `res/layout` folder.

Investigate the XML layout in the visual editor as well as the XML structure.

### 23.2. Remove exiting views

Remove all views, except the top level entry which is the layout manager. In the visual design mode you can remove a view by right-clicking it and by selecting the *Delete* entry for the context menu.

The result layout file should look similar to the following file.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

</RelativeLayout>
```

### 23.3. Add and configure views

Add an `EditText` (Plain Text) and a `Button` to your layout. The easiest way is to find these elements in the *Palette* and drag and drop them into your layout.


Use the XML editor to change the ID of the new `EditText` field to *main\_input*. In the XML file this looks like `@+id/main_input`.

Change the background color of your layout to silver (`#C0C0C0`). Use the `android:background` property for this.

### 23.4. Add interactive button

Change the button text to *Start* via the `android:text` property in your layout file.

Assign the name *onClick* to the `android:onClick` property of your `Button`.

**Tip**

This defines that a public void `onClick (View view)` method is be called in the activity once the button is pressed.

The resulting layout should look like the following screenshot.



Implement the following method in your MainActivity class.

```
public void onClick (View view) {  
    Toast.makeText(this, "Button 1 pressed",  
        Toast.LENGTH_LONG).show();  
}
```

## 23.5. Validate button interaction

Start your application. Press your button and validate that a popup message (Toast) is shown.

## 23.6. Display text from your EditText field

Go back to the source code and use the `findViewById(id)` method with the correct `id` and cast the returned object into `EditText`, e.g. `EditText text = (EditText) findViewById(id)`. You can get the right `id` via the `R` class. It should

be stored under ID and called *main\_input*.

Use the `text.getText().toString()` method to read the string in the editor field and add the text to your Toast message.

## 23.7. Validate popup message

Restart your application and ensure that the `Toast` displays the text which the `EditText` field contains.

## 23.8. Solution

After these changes your layout file should be similar to the following code.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/main_input"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="14dp"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="31dp"
        android:onClick="onClick"
        android:text="Start" />

</RelativeLayout>
```

And your activity coding should look similar to the following code.

```

package com.vogella.android.first;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (BuildConfig.DEBUG) {
            Log.d(Constants.LOG, "onCreated called");
        }
        setContentView(R.layout.activity_main);
    }

    // you may have here an onCreateOptionsMenu method
    // this method is not required for this exercise
    // therefore I deleted it

    public void onClick(View view) {
        EditText input = (EditText) findViewById(R.id.main_input);
        String string = input.getText().toString();
        Toast.makeText(this, string, Toast.LENGTH_LONG).show();
    }
}

```

## 24. Exercise: Influence view layout at runtime

### 24.1. Add radio group and radio buttons to your layout

Continue to use the project called `com.vogella.android.first`. In this exercise you add radio buttons your layout. Depending on the user selection the radio button arrangement changes from horizontal to vertical.

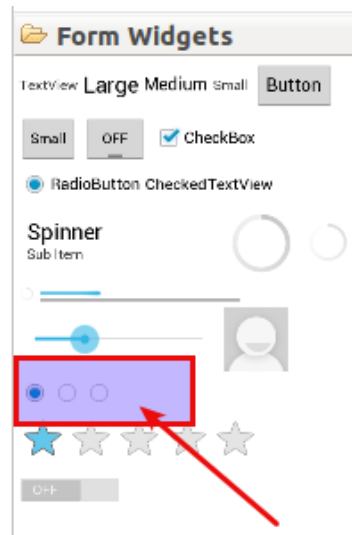
Open your layout file and add a radio group with two radio buttons to your layout.





In case you have problems with the creation of the radio buttons with the visual editor, you find the resulting XML snippet later in this exercise.

The radio group is hard to find in the *Palette* view. The widget is highlighted in the following screenshot. If required, remove radio buttons until you have only two buttons.



### Tip

To assign IDs to views, use the `android:id` attribute, for example:  
`android:id="@+id/orientation`.

Assign them based on the following table.

**Table 7. ID Assignment**

ID	View
orientation	Radio Group
horizontal	First radio button

vertical

Second radio button

The resulting part of the layout file should be similar to the following listing.

```
<!-- this snippet is part of the larger layout file -->

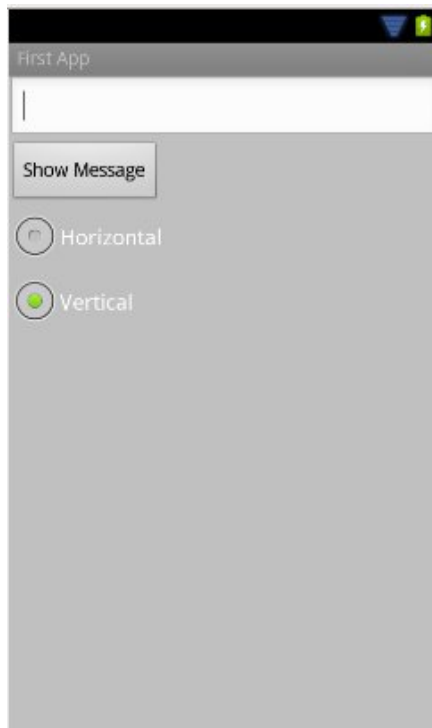
<RadioGroup
    android:id="@+id/orientation"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <RadioButton
        android:id="@+id/horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Horizontal" >
    </RadioButton>

    <RadioButton
        android:id="@+id/vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="Vertical" >
    </RadioButton>
</RadioGroup>
```

The resulting layout should look like the following screenshot.





## 24.2. Change radio group orientation dynamically

Change the `onCreate()` method in your activity. Use the `findViewById()` method to find the `RadioGroup` in your layout.

Implement a listener on the radio group which changes the orientation of the radio buttons based on the current selection of the buttons. Which button is selected, can be identified by the ID parameter.



### Tip

`RadioGroup` allows you to add a `RadioGroup.OnCheckedChangeListener` from the `android.widget.RadioGroup` package via the `setOnCheckedChangeListener()` method. This listener is notified if the selection of the radio group changes.

You can use the following code as template to implement the listener.

```
RadioGroup group1 = (RadioGroup) findViewById(R.id.orientation)
;
group1.setOnCheckedChangeListener(new RadioGroup.OnCheckedChange
eListener() {
```

```
@Override
public void onCheckedChanged(RadioGroup group, int checkedId)
{
    switch (checkedId) {
        case R.id.horizontal:
            group.setOrientation(LinearLayout.HORIZONTAL);
            break;
        case R.id.vertical:
            group.setOrientation(LinearLayout.VERTICAL);
            break;
    }
}
});
```

### 24.3. Validate

Run your application and select the different radio button. Ensure that the orientation of the buttons is changed based on your selection.

## 25. Exercise: Create a temperature converter

### 25.1. Demo application

In this exercise you learn how to create and consume Android resources and repeat the creation of an interactive application.

This application is available on Google Play under the following URL: [Android Temperature converter](#)

Alternatively you can also scan the following barcode with your Android phone to install it via the Google Play application.



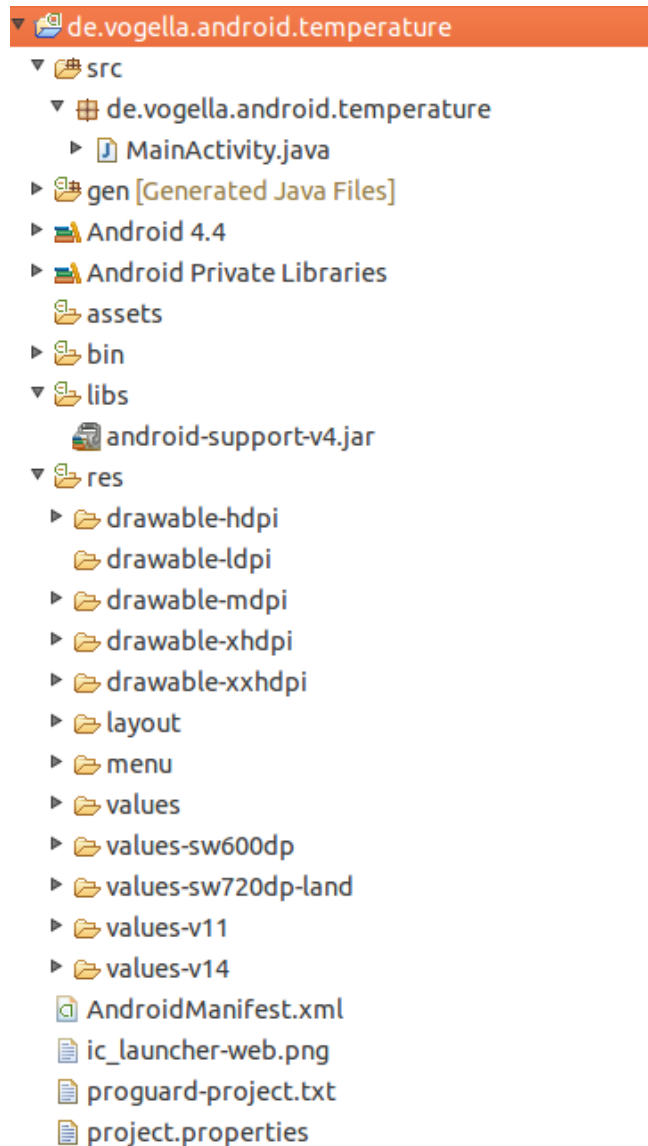
## 25.2. Create Project

Select *File* → *New* → *Other...* → *Android* → *Android Application Project* to create a new Android project with the following data.

**Table 8. New Android project**

Property	Value
Application Name	Temperature Converter
Project Name	de.vogella.android.temperature
Package name	de.vogella.android.temperature
API (Minimum, Target, Compile with)	Latest
Template	Empty Activity
Activity	MainActivity
Layout	activity_main

After the wizard ends, a project structure similar to the following picture is created.

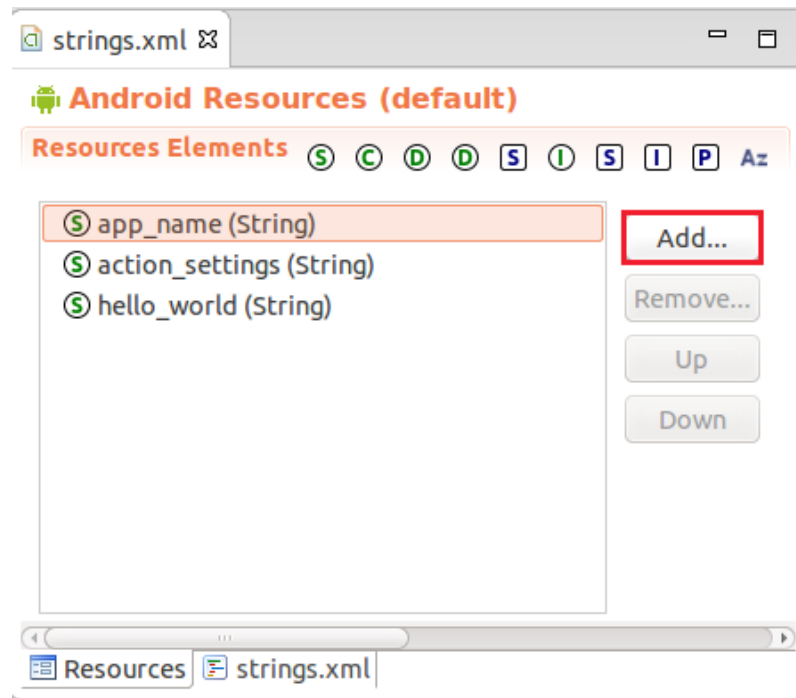


## 25.3. Create attributes

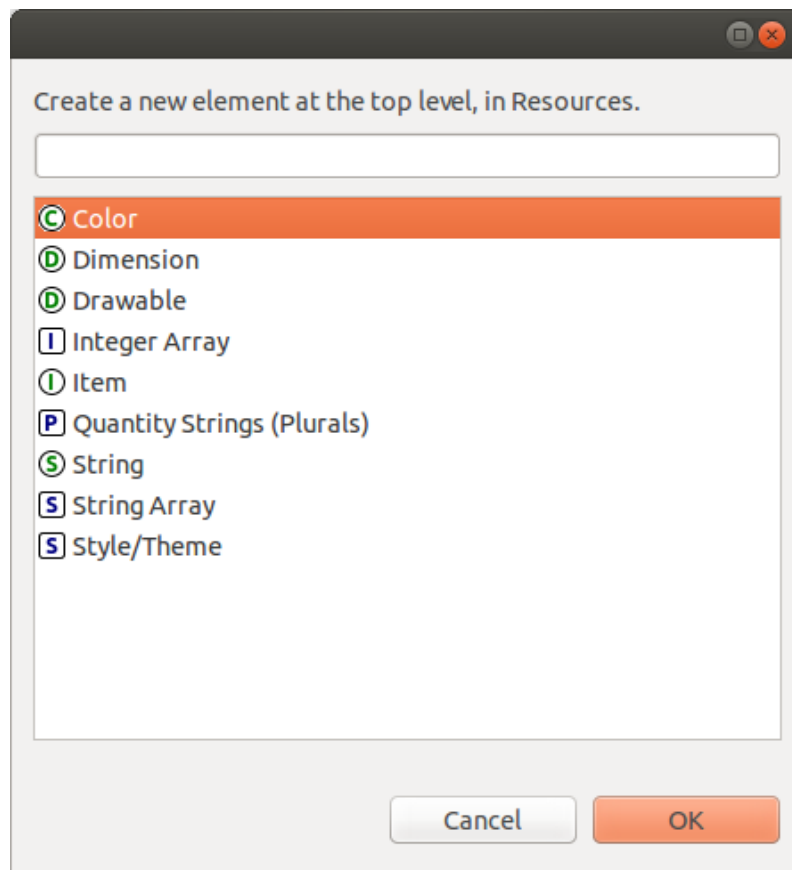
Android allows you to create static resources to define attributes, e.g., Strings or colors. These attributes can be used in other XML files or by Java source code.

Select the *res/values/string.xml* file to open the editor for this file.

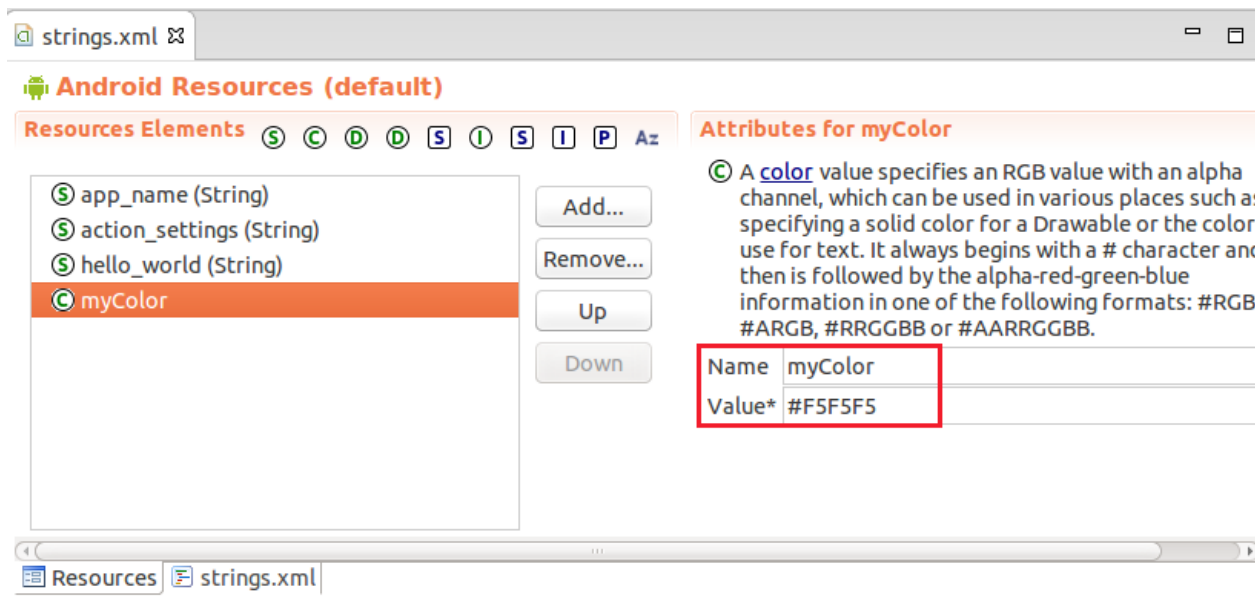
You want to add a `color` definition to the file. Press the *Add* button for this.



Select the *Color* entry in the following dialog and press the *OK* button.



Enter *myColor* as the name and *#F5F5F5* as the value.



Add more attributes, this time of the *String* type. String attributes allow the developer to translate the application at a later point.

**Table 9. String Attributes**

Name	Value
celsius	to Celsius
fahrenheit	to Fahrenheit
calc	Calculate

Switch to the XML representation and validate that the values are correct.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Temperature Converter</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <color name="myColor">#F5F5F5</color>
    <string name="celsius">to Celsius</string>
```

```
<string name="fahrenheit">to Fahrenheit</string>
<string name="calc">Calculate</string>

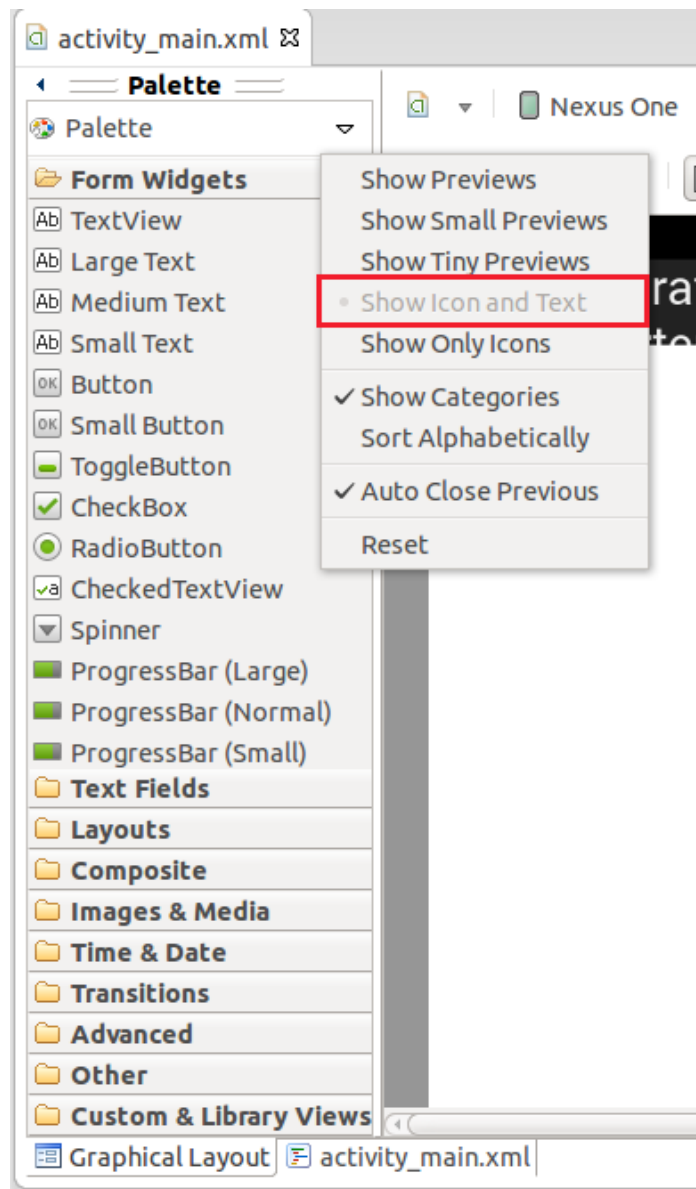
</resources>
```

## 25.4. Using the layout editor

Select the *res/layout/activity\_main.xml* file. Open the associated Android editor via a double-click on the file. This editor allows you to create the layout via drag and drop or via the XML source code. You can switch between both representations via the tabs at the bottom of the editor. For changing the position and grouping elements you can use the Eclipse *Outline* view.

The following shows a screenshot of the *Palette* side of this editor. This element allows you to drag and drop new *view* elements into your layout. To identify the *view* elements easier, you can switch the representation to show the icon and the text as depicted in the following screenshot.





### Note

The *Palette* view changes frequently so your view might be a bit different.

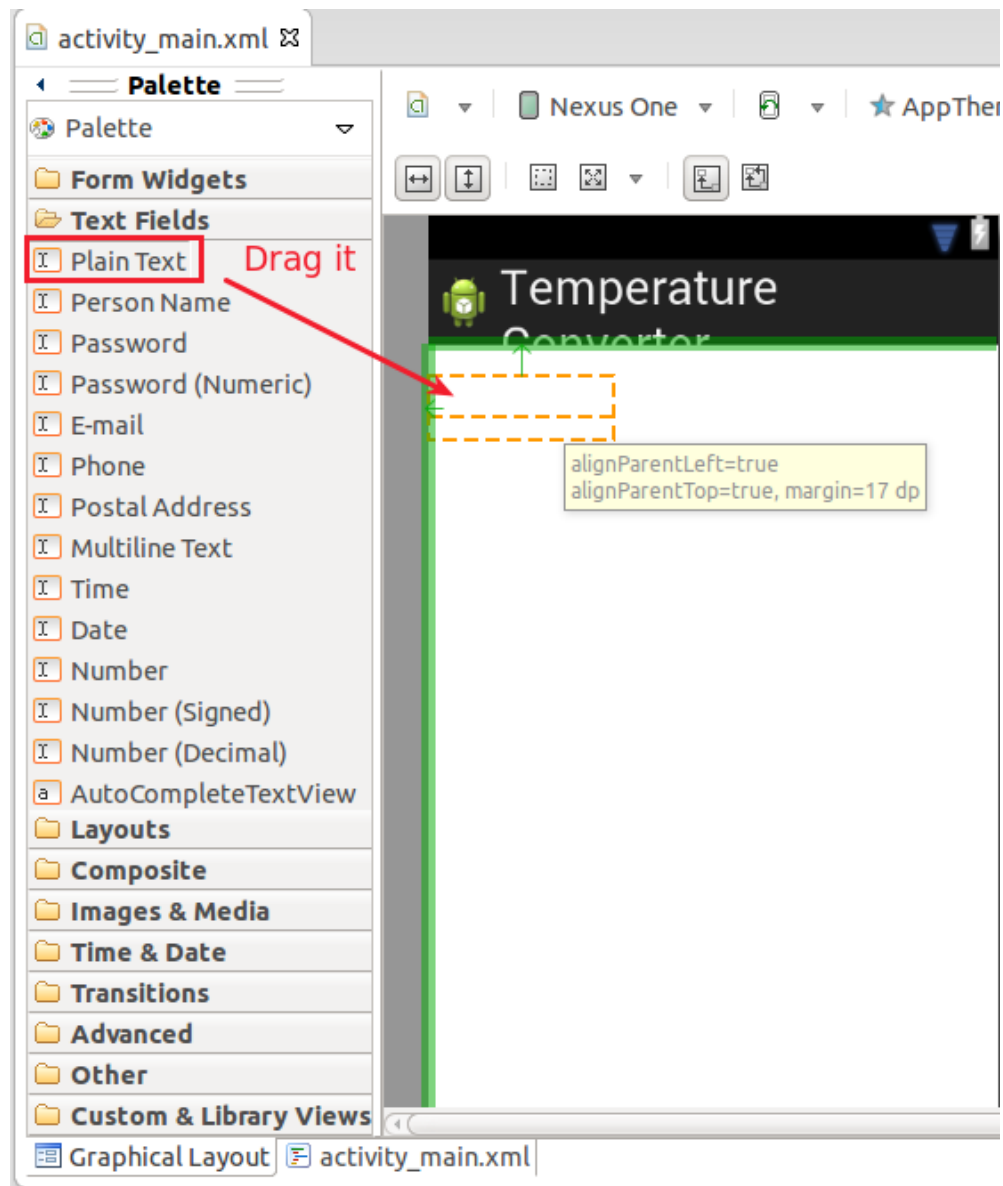
## 25.5. Add views to your layout file

In this part of the exercise you create the base user interface for your application.

Right-click on the existing *Hello World!* text object in the layout. Select *Delete* from the popup menu to remove the text object.

Afterwards select the *Text Fields* section in the *Palette* and locate the *Plain Text* (via the tooltip).

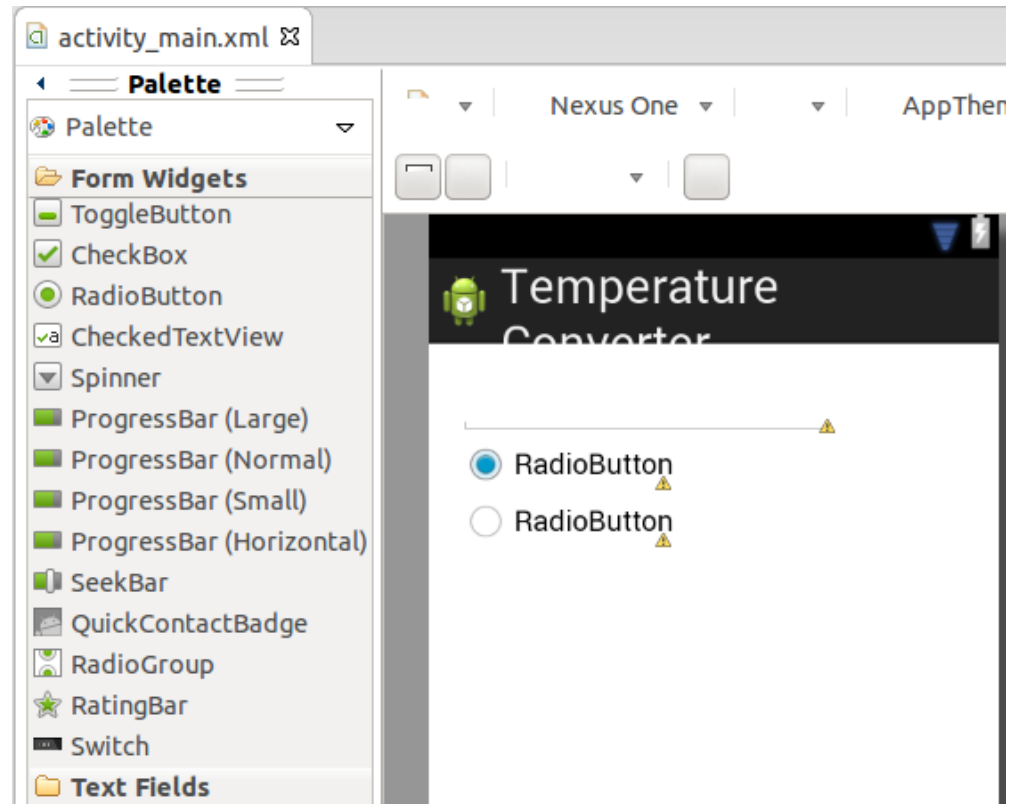
Click on the *Text Fields* section header to see all text fields. Drag the *Plain Text* widget onto your layout to create a text input field.



### Note

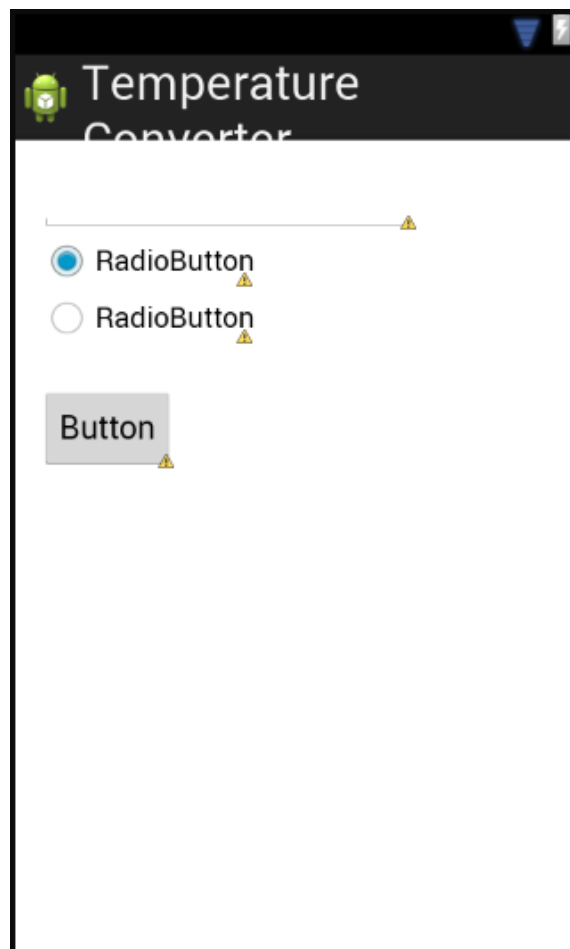
All entries in the *Text Fields* section define text fields. The different entries define additional attributes for them, e.g., if a text field should only contain numbers.

Afterwards select the *Form Widgets* section in the *Palette* and drag a *RadioGroup* entry into the layout. The number of radio buttons added to the radio button group depends on your version of Eclipse. Make sure there are two radio buttons by deleting or adding radio buttons to the group.



Drag a Button from the *Form Widgets* section into the layout.

The result should look like the following screenshot.



Switch to the XML tab of your layout file and verify that the file looks similar to the following listing. ADT changes the templates from time to time, so your XML might look slightly different.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:ems="10" />

<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/editText1" >

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="RadioButton" />

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RadioButton" />
</RadioGroup>

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/radioGroup1"
    android:layout_below="@+id/radioGroup1"
    android:layout_marginTop="22dp"
    android:text="Button" />

</RelativeLayout>

```



### Note

You see some warning messages because you use hard-coded Strings. You will fix this in the next section of this exercise.

## 25.6. Edit view properties

You can add and change properties of a view directly in the XML file. The Android tooling also provides content assists via the Ctrl+Space shortcut.



## Tip

The attributes of a view can also be changed via the Eclipse *Properties* view or via the context menu of the view. But changing properties in the XML file is typically faster if you know what you want to change.

Switch to the XML file and assign the `@string/celsius` value to the `android:text` property of the first radio button. Assign the `fahrenheit` string attribute to the `text` property of the second radio button.

```
<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/editText1" >

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="@string/celsius" />

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/fahrenheit" />
</RadioGroup>
```



## Note

From now on this description assumes that you are able to modify the properties of your views in a layout file.

Ensure that the *Checked* property is set to `true` for the first `RadioButton`.

Assign `@string/calculator` to the text property of your button and assign the value `onClick` to the `onClick` property.

Set the `inputType` property to `numberSigned` and `numberDecimal` on the `EditText`. As an example you can use the last line in the following XML snippet.

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:ems="10"
    android:inputType="numberSigned|numberDecimal" />
```

All your user interface components are contained in a layout. Assign the background color to this `Layout`. Select `Color` and then select `myColor` in the dialog. As an example you can use the last line in the following XML snippet.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="@color/myColor">
```

Afterwards the background should change to the `whitesmoke` color. It might be difficult to see the difference.

Switch to the `activity_main.xml` tab and verify that the XML is correct.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity"
    android:background="@color/myColor">

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```



```

        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:ems="10"
        android:inputType="numberSigned|numberDecimal" />

<RadioGroup
    android:id="@+id/radioGroup1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_below="@+id/editText1" >

    <RadioButton
        android:id="@+id/radio0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="@string/celsius" />

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/fahrenheit" />
</RadioGroup>

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/radioGroup1"
    android:layout_below="@+id/radioGroup1"
    android:layout_marginTop="22dp"
    android:text="@string/calc"
    android:onClick="onClick" />

</RelativeLayout>

```

## 25.7. Create utility class

Create the following utility class to convert from celsius to fahrenheit and vice versa.

```

package de.vogella.android.temperature;

public class ConverterUtil {
    // converts to celsius
    public static float convertFahrenheitToCelsius(float fahrenheit) {
        return ((fahrenheit - 32) * 5 / 9);
    }
}

```

```
// converts to fahrenheit
public static float convertCelsiusToFahrenheit(float celsius) {
    return ((celsius * 9) / 5) + 32;
}
}
```

## 25.8. Change the activity code

The Android project wizard created the corresponding `MainActivity` class for your activity code. Adjust this class to the following.

```
package de.vogella.android.temperature;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends Activity {
    private EditText text;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text = (EditText) findViewById(R.id.editText1);
    }

    // this method is called at button click because we assigned the name to the
    // "OnClick property" of the button
    public void onClick(View view){
        switch (view.getId()) {
            case R.id.button1:
                RadioButton celsiusButton = (RadioButton) findViewById(R.id.radio0);
                RadioButton fahrenheitButton = (RadioButton) findViewById(R.id.radio1);
                if (text.getText().length() == 0) {
                    Toast.makeText(this, "Please enter a valid number",
                        Toast.LENGTH_LONG).show();
                    return;
                }

                float inputValue = Float.parseFloat(text.getText().toString());
                if (celsiusButton.isChecked()) {
                    text.setText(String
                        .valueOf(ConverterUtil.convertFahrenheitToCelsius(inputValue)));
                    celsiusButton.setChecked(false);
                    fahrenheitButton.setChecked(true);
                }
            }
        }
    }
}
```

```
    } else {  
        text.setText(String  
            .valueOf(ConverterUtil.convertCelsiusToFahrenheit(inputValue)));  
        fahrenheitButton.setChecked(false);  
        celsiusButton.setChecked(true);  
    }  
    break;  
}  
}
```



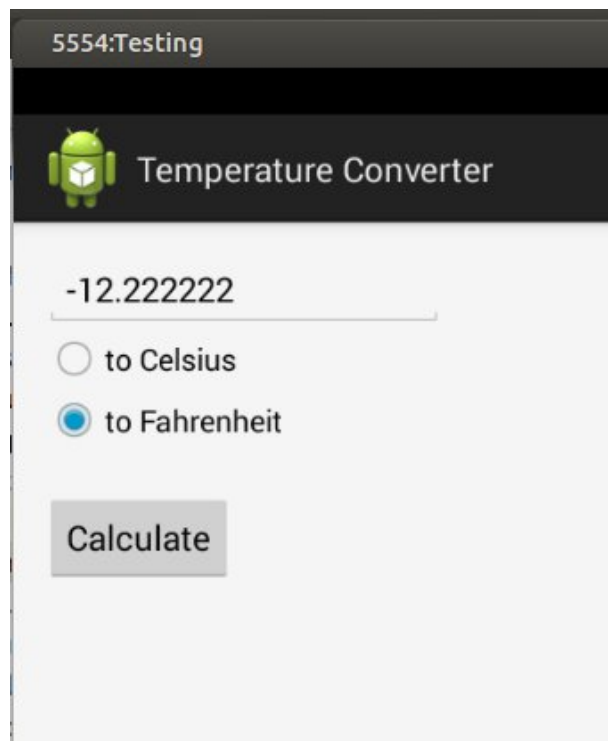
### Note

The `onClick` is called by a click on the button because of the `onClick` property of the button.

## 25.9. Start application

Right click on your project and select *Run-As → Android Application*. If an emulator is not yet running, it will be started.

Type in a number, select your conversion and press the button. The result should be displayed and the other option should get selected.



## 26. Using Resources

### 26.1. References to resources in code

The `Resources` class allows to access individual resources. An instance of the `Resources` class can be retrieved via the `getResources()` method of the `Context` class. As activities and services extend the `Context` class, you can directly use this method in implementations of these components.

An instance of the `Resources` class is also required by other Android framework classes. For example, the following code shows how to create a `Bitmap` file from a reference ID.

```
// BitmapFactory requires an instance of the Resource class
BitmapFactory.decodeResource(getResources(), R.drawable.ic_action_search);
```

### 26.2. Accessing views from the layout in an activity

In your activity (and fragment) code you frequently need to access the views to access and modify their properties.

In an activity you can use the `findViewById(id)` method call to search for a view in the current layout. The *id* is the ID attribute of the view in the layout. The usage of this method is demonstrated by the following code.

```
package com.vogella.android.first;

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

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView textView = (TextView) findViewById(R.id.mytext);

        // TODO do something with the TextView
    }
}
```

It is also possible to search in a view hierarchy with the `findViewById(id)` method, as demonstrated in the following code snippet.

```
// search in the layout of the activity
LinearLayout linearLayout = (LinearLayout) findViewById(R.id.mylayout);

// afterwards search in linearLayout for another view
TextView textView = (TextView) linearLayout.findViewById(R.id.mytext);

// note, you could have directly searched for R.id.mytext, the above coding
// is just for demonstration purposes
```

## 26.3. Reference to resources in XML files

In your XML files, for example, your layout files, you can refer to other resources via the `@` sign.

For example, if you want to refer to a color, which is defined in an XML resource, you can refer to it via `@color/your_id`. Or if you defined a string with the "titlepage" key in an XML resource, you could access it via `@string/titlepage`.

## 26.4. Reference to Android system resources in XML files

To use an Android system resource, include the `android` namespace into the references, e.g.,

## 27. Assets

### 27.1. Whats are assets?

While the *res* directory contains structured values which are known to the Android platform, the *assets* directory can be used to store any kind of data.

You can access files stored in this folder based on their path. The *assets* directory also allows you to have sub-folders.



#### Note

You could also store unstructured data in the */res/raw* folder, but it is considered good practice to use the *assets* directory for such data.

### 27.2. Accessing assets

You access this data via the `AssetsManager` which you can access via the `getAssets()` method from an instance of the `Context` class.

The `AssetsManager` class allows you to read a file in the *assets* folder as `InputStream` with the `open()` method. The following code shows an example for this.

```
// get the AssetManager
AssetManager manager = getAssets();

// read the "logo.png" bitmap from the assets folder
InputStream open = null;
try {
    open = manager.open("logo.png");
    Bitmap bitmap = BitmapFactory.decodeStream(open);
    // assign the bitmap to an ImageView in this layout
    ImageView view = (ImageView) findViewById(R.id.imageView1);
    view.setImageBitmap(bitmap);
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (open != null) {
        try {
```

```
open.close();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

## 28. Exercise: Using resources in XML files and in code

### 28.1. Add images to your project

Continue to use the project `com.vogella.android.first`.

Place two `.png` files of your choice in the `/res/drawable-mdpi` folder. The first image should be called "initial.png" and the second should be called "assigned.png".



#### Tip

If you don't have any `.png` files at hand, perform a Google search for "Android png files".

### 28.2. Add views to your project

Open your layout file and add a new `Button` and an `ImageView` to it.

### 28.3. Assign image to your image view

Assign the `initial.png` file to your `ImageView`, via your layout file as demonstrated in the following XML snippet.

```
<!--
    NOTE: More attributes are required
    for the correct layout of the ImageView. These are left
    out for brevity
-->

<ImageView
    android:id="@+id/myicon"
```

```
.... more attributes
android:src="@drawable/initial" />
```

## 28.4. Replace images via button click

If the button is clicked, use the `findViewById()` to search for the `ImageView`. Use its `setImageResource()` method to assign the png file (which is represented at runtime via a `Drawable` object) to your `ImageView`.



### Tip

The parameter of the `setImageResource()` method is the `R` reference to your file, e.g., `R.drawable.your_png_file`.

## 28.5. Validate

Ensure that if you press your new button, the displayed image is replaced.

# 29. Exercise: Using ScrollView

This exercise demonstrates the usage of the `ScrollView` view to provide a scrollable user interface component. Create an android project *de.vogella.android.scrollview* with the activity called *ScrollViewActivity*. Use *activity\_main.xml* as layout file.

Change the layout file used in the activity to the following.

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

    <LinearLayout
        android:id="@+id/LinearLayout01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/TextView01"
```



```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="8dip"
        android:paddingRight="8dip"
        android:paddingTop="8dip"
        android:text="This is a header"
        android:textAppearance="?android:attr/textAppearanceLarge" >
    </TextView>

    <TextView
        android:id="@+id/TextView02"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1.0"
        android:text="@+id/TextView02" >
    </TextView>

    <LinearLayout
        android:id="@+id/LinearLayout02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1.0"
            android:text="Submit" >
        </Button>

        <Button
            android:id="@+id/Button02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1.0"
            android:text="Cancel" >
        </Button>
    </LinearLayout>
</LinearLayout>

</ScrollView>

```

Change your `ScrollViewActivity` class to the following code.

```

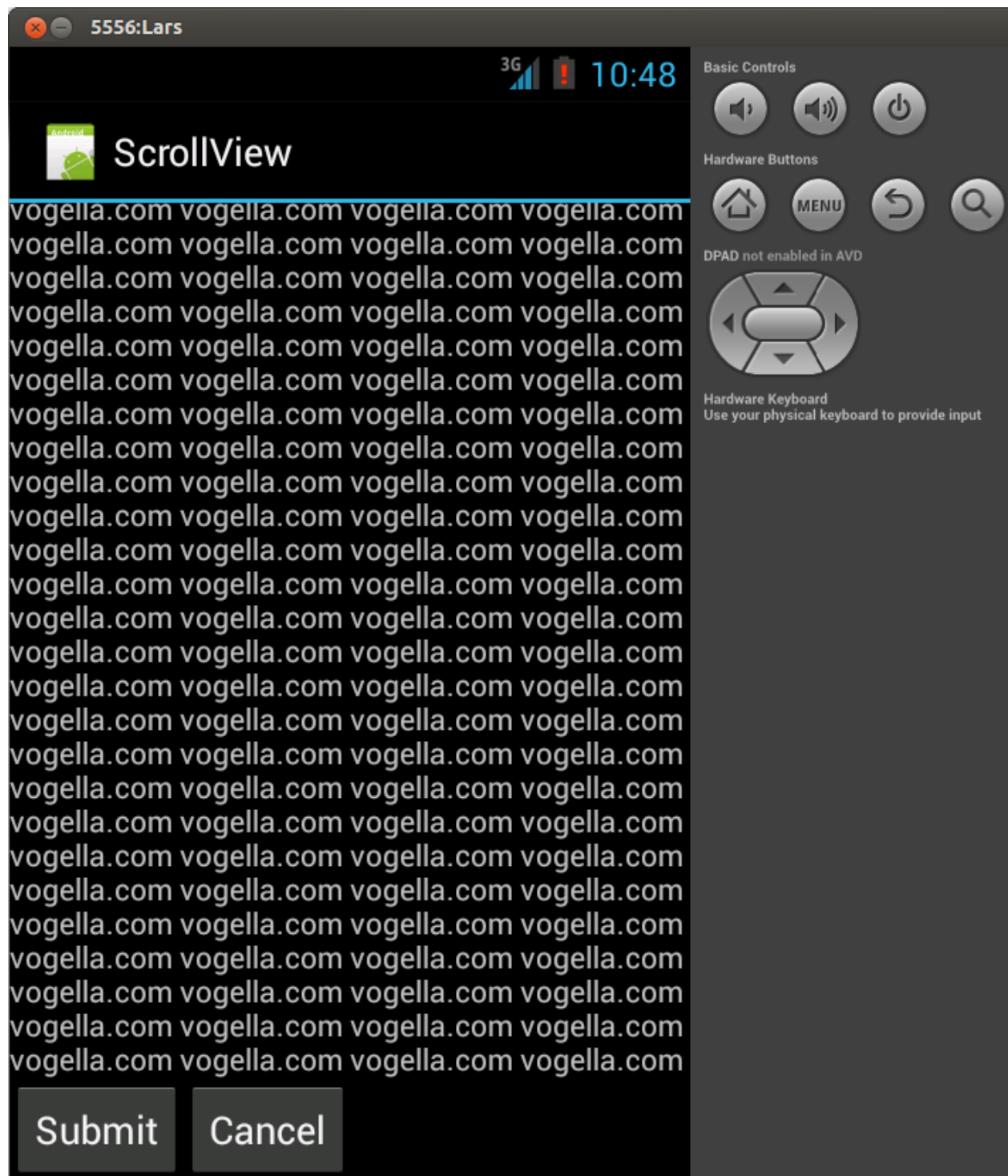
package de.vogella.android.scrollview;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

```

```
public class ScrollViewActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        TextView view = (TextView) findViewById(R.id.TextView02);  
        String s="";  
        for (int i=0; i < 500; i++) {  
            s += "vogella.com ";  
        }  
        view.setText(s);  
    }  
}
```

Start your application and ensure that you can scroll down to the buttons.



# 30. Deployment

## 30.1. Overview

In general there are restrictions how to deploy an Android application to your device. You can use USB, email yourself the application or use one of the many Android markets to install the application. The following description highlights the most common ones.

## 30.2. Deployment via Eclipse

Turn on *USB Debugging* on your device in the settings. Select *Settings* → *Development Options*, then enable the *USB-Debugging* option.

You may also need to install the driver for your mobile phone. Linux and Mac OS usually work out of the box while Windows typically requires the installation of a driver.

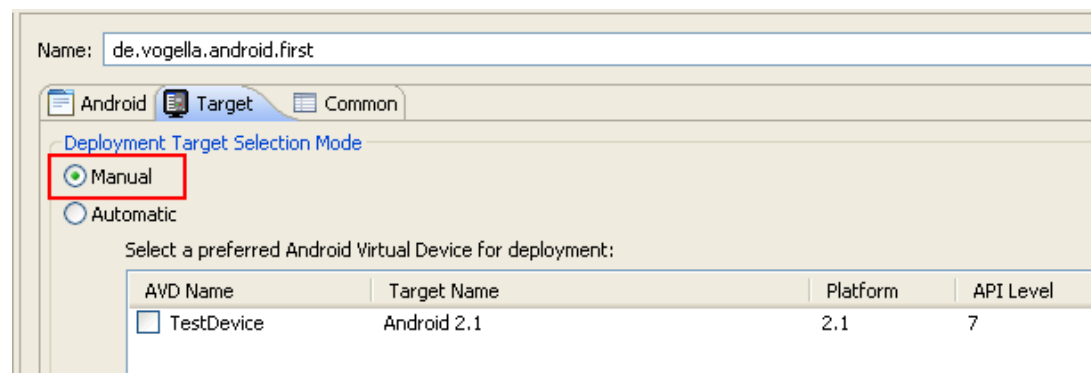
For details on the driver installation on Windows please see [Google guide for device deployment](#).

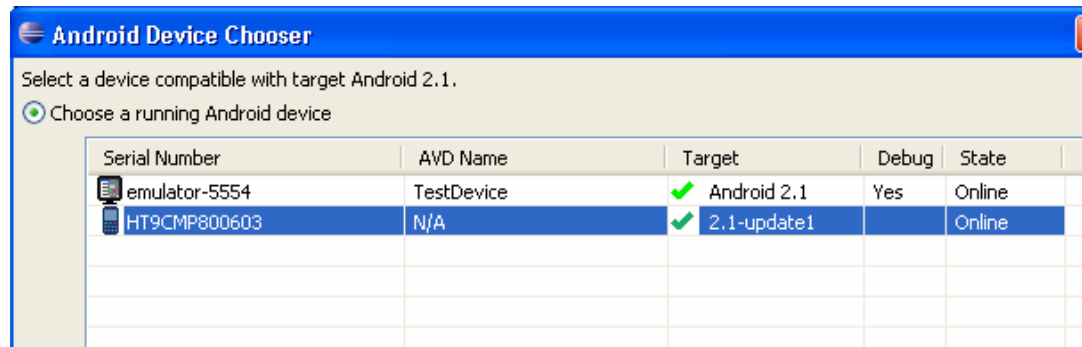


### Note

The minimum Android version of your Android application needs to fit to the Android version on your device.

If you have several devices connected to your computer, you can select which one should be used. If only one device is connected, the application is automatically deployed on this device.





### 30.3. Export your application

Android applications must be signed before they can get installed on an Android device. During development Eclipse signs your application automatically with a debug key.

If you want to install your application without the Eclipse IDE, you can right-click on it and select *Android Tools* → *Export Signed Application Package*.

This wizard allows to use an existing key or to create a new one.

Please note that you need to use the same signature key in Google Play (Google Market) to update your application. If you lose the key, you will NOT be able to update your application ever again.

Make sure to backup your key.

### 30.4. Via external sources

Android also allows to install applications directly. Just click on a link which points to an *.apk* file, e.g., in an email attachment or on a webpage. Android will prompt you if you want to install this application.

This requires a setting on the Android device which allows the installation of non-market application. Typically this setting can be found under the "Security" settings.

### 30.5. Google Play (Market)

Google Play requires a one time fee, currently 25 Dollar. After that the developer can directly upload his application and the required icons, under **Google Play Publishing**.

Google performs some automatic scanning of applications, but no approval process is in place. All application, which do not contain malware, will be published. Usually a few minutes after upload, the application is available.



## 31. Support free vogella tutorials

Maintaining high quality free online tutorials is a lot of work. Please support free tutorials by donating or by reporting typos and factual errors.

### 31.1. Thank you

Please consider a contribution if this article helped you.



### 31.2. Questions and Discussion

If you find errors in this tutorial, please notify me (see the [top of the page](#)). Please note that due to the high volume of feedback I receive, I cannot answer questions to your implementation. Ensure you have read the [vogella FAQ](#) as I don't respond to questions already answered there.



## 32. Links and Literature

## 32.1. Source Code

[Source Code of Examples](#)

## 32.2. Android Online Resources

[Android Developer Homepage](#)

[vogella Android online tutorials](#)

[Android Issues / Bugs](#)

[Android Google Groups](#)

[Android emulator skins](#)

## 32.3. vogella Resources

[vogella Training](#) Android and Eclipse Training from the vogella team

[Android Tutorial](#) Introduction to Android Programming

[GWT Tutorial](#) Program in Java, compile to JavaScript and HTML

[Eclipse RCP Tutorial](#) Create native applications in Java

[JUnit Tutorial](#) Test your application

[Git Tutorial](#) Put all your files in a distributed version control system