

3.2k

Free tutorial, donate
to support




by Lars Vogel



Android Development Tutorial

Based on Android 4.2

Lars Vogel

Version 11.2

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

20.01.2013

Revision History

Revision 0.1	04.07.2009	Lars Vogel	Created
Revision 0.2 - 11.2	07.07.2009 - 20.01.2013		bug fixing and enhancements

Development with Android and Eclipse

This tutorial describes how to create Android applications with Eclipse. It is based on Eclipse 4.2 (Juno), Java 1.6 and Android 4.2 (Jelly Bean).

Table of Contents

1. What is Android?
 - 1.1. Android Operation System
 - 1.2. Google Play (Android Market)
2. Security and permissions
 - 2.1. Security concept in Android
 - 2.2. Permission concept in Android
3. Android applications and tasks
 - 3.1. Application
 - 3.2. Tasks across application borders
4. Android user interface components
 - 4.1. Activity
 - 4.2. Fragments
 - 4.3. Views and layout manager
 - 4.4. Device configuration specific layouts
5. Other Android components
 - 5.1. Intents
 - 5.2. Services
 - 5.3. ContentProvider
 - 5.4. BroadcastReceiver
 - 5.5. (HomeScreen) Widgets
 - 5.6. Live Wallpapers
6. Android Development Tools
 - 6.1. Android SDK
 - 6.2. Android Development Tools
 - 6.3. Dalvik Virtual Machine
 - 6.4. How to develop Android Applications
 - 6.5. Resource editors
7. Android Application Architecture
 - 7.1. AndroidManifest.xml
 - 7.2. Activities and Lifecycle
 - 7.3. Configuration Change
 - 7.4. Context



- 8.1. Support in Android for resource files
- 8.2. Defining IDs
- 9. Using Resources
 - 9.1. Reference to resources in code
 - 9.2. Reference to resources in XML files
 - 9.3. Activities and Layouts
- 10. Assets
 - 10.1. Whats are assets?
 - 10.2. Accessing assets
- 11. Installation
 - 11.1. Options
 - 11.2. Standalone ADT installation
- 12. Android virtual device - Emulator
 - 12.1. What is the Android Emulator?
 - 12.2. Google vs. Android AVD
 - 12.3. Emulator Shortcuts
 - 12.4. Parameter
- 13. Tutorial: Create and run Android Virtual Device
 - 13.1. Create AVD
 - 13.2. Run AVD
 - 13.3. Stopping the emulator
- 14. Solving Android development problems
- 15. Conventions and API level
 - 15.1. API version
 - 15.2. Android project and package name
- 16. Tutorial: create and run Android project
- 17. Views
 - 17.1. Available widgets in Android
 - 17.2. View class
- 18. Tutorial: Create a temperature converter
 - 18.1. Install the demo application
 - 18.2. Create Project
 - 18.3. Create attributes
 - 18.4. Add Views
 - 18.5. Edit view properties
 - 18.6. Change the Activity source code
 - 18.7. Start Project
- 19. Starting an installed application
- 20. Layout Manager and ViewGroups
 - 20.1. Available Layout Manager
 - 20.2. FrameLayout
 - 20.3. LinearLayout
 - 20.4. RelativeLayout
 - 20.5. GridLayout
 - 20.6. ScrollView
- 21. Tutorial: ScrollView
- 22. DDMS perspective and important views
 - 22.1. DDMS - Dalvik Debug Monitor Server
 - 22.2. LogCat View
 - 22.3. File explorer
- 23. Deployment
 - 23.1. Overview
 - 23.2. Deployment via Eclipse
 - 23.3. Export your application
 - 23.4. Via external sources
 - 23.5. Google Play (Market)
- 24. Thank you
- 25. Questions and Discussion
- 26. Links and Literature
 - 26.1. Source Code
 - 26.2. Android Resources
 - 26.3. vogella Resources

1. What is Android?

1.1. Android Operation System

Android is an operating system based on Linux with a Java programming interface.

The Android Software Development Kit (Android SDK) provides all necessary tools to develop Android applications. This includes a compiler, debugger and a device emulator, as well as its own virtual machine to run Android programs.

Android is currently primarily developed by Google.

Android allows background processing, provides a rich user interface library, supports 2-D and 3-D graphics using the OpenGL libraries, access to the file system and provides an embedded SQLite database.

Android applications consist of different components and can re-use components of other applications. This leads to the concept of a *task* in Android; an application can re-use other Android components to archive a task. For example you can trigger from your application another application which has itself registered with the Android system to handle photos. In this other application you select a photo and return to your application to use the selected photo.

1.2. Google Play (Android Market)

Google offers the *Google Play* service in which programmers can offer their Android application to Android users. Google phones include the *Google Play* application which allows to install applications.

Google Play also offers an update service, e.g. if a programmer uploads a new version of his application to Google Play, this service will notify existing users that an update is available and allow to install it.

Google Play used to be called *Android Market*.

2. Security and permissions

2.1. Security concept in Android

During deployment on an Android device, the Android system will create a unique user and group ID for every Android application. Each application file is private to this generated user, e.g. other applications cannot access these files.

In addition each Android application will be started in its own process.

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

If data should be shared, the application must do this explicitly, e.g. via a *service* or a *ContentProvider*.

2.2. Permission concept in Android

Android also contains a permission system. Android predefines permissions for certain tasks but every application can define additional permissions.

An Android application declare its required permissions in its *AndroidManifest.xml* configuration file. 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,

In most cases the requested permissions will be presented to the user before installation of the application. The user needs to decide if these permissions are given to the application.

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

Not all users pay attention to the required permissions during installation. But some users do and they write negative reviews on Google Play.

3. Android applications and tasks

3.1. Application

An Android application consists out of different Android components and additional resources. The Android system knows *activities*, *services*, *broadcast receiver* and *content provider* as components.

3.2. Tasks across application borders

Android application components can connect to components of other Android applications to create *tasks*. For example an application which allows you to make a photo can start an email application and instruct this application to create a new email and attach a photo to this email.

4. Android user interface components

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

4.1. Activity

An *activity* represents the visual representation of an Android application. *activities* use *views*, i.e. user interface widgets as for example buttons and *fragments* to create the user interface and to interact with the user.

An Android application can have several *activities*.

4.2. Fragments

Fragments are components which run in the context of an *Activity*. A *Fragment* encapsulates application code so that it is easier to reuse it and to support different sized devices.

Fragments are optional components which allow you to reuse user interface and non user interface components for different devices configurations.

4.3. Views and layout manager

Views are user interface widgets, e.g. buttons or text fields. The base class for all *views* is the `android.view.View` class. *Views* have attributes which can be used to configure their appearance and behavior.

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

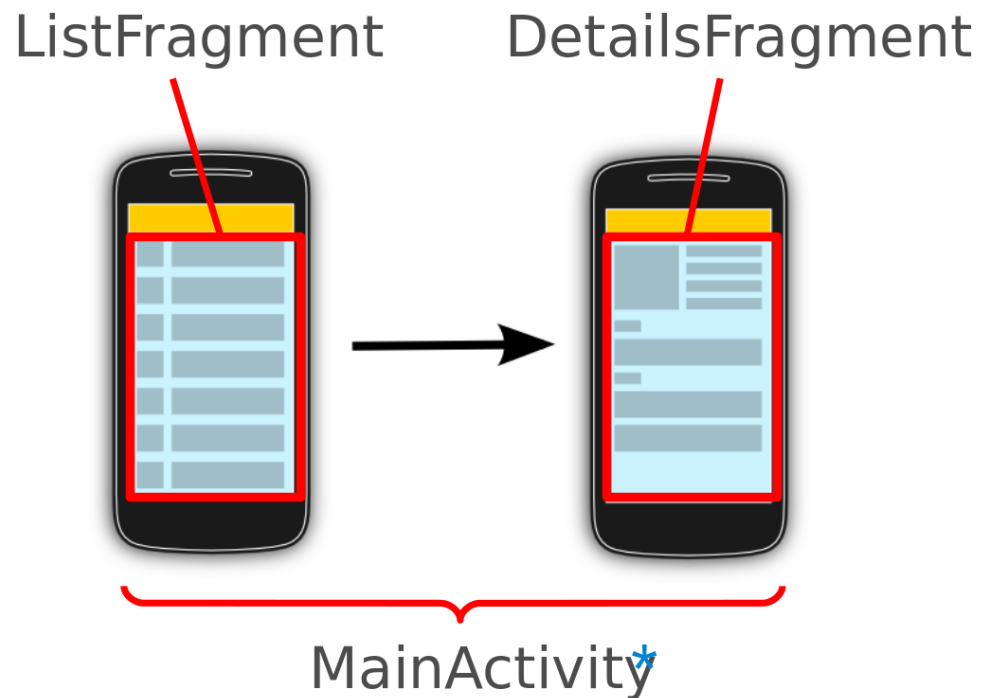
Layout managers can be nested to create complex layouts. You should avoid nesting them too deeply.

4.4. Device configuration specific layouts

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

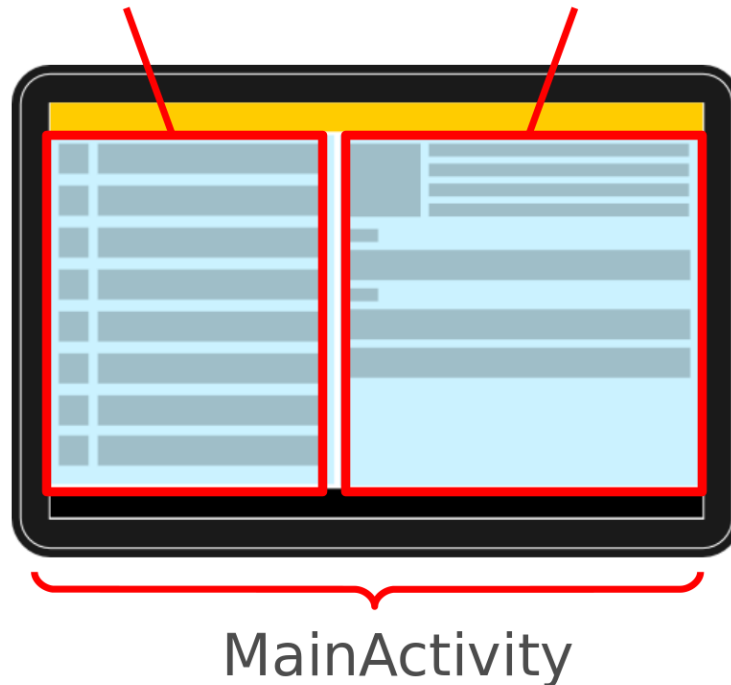
Fragments are designed to support such a setup.

The following pictures shows an *activity* called *MainActivity*. On a smaller screen it shows one *Fragment* and allows that the user navigates to another *Fragment*. On a wide screen it shows two *Fragments*.



ListFragment

DetailsFragment



5. Other Android components

Android has several more components which can be used in your Android application.

5.1. Intents

Intents are asynchronous messages which allow the application to request functionality from other Android components, e.g. from *services* or *activities*.

An application can call a component directly (*explicit Intent*) or ask the Android system to evaluate registered components based on the *intent* data (*implicit intents*). For example the application could implement sharing of data via an *intent* and all components which allow sharing of data would be available for the user to select. Applications register themselves to an *intent* via an *intentFilter*.

Intents allow an Android application to start and to interact with components from other Android applications.

5.2. Services

Services perform tasks without providing a user interface. They can communicate with other Android components and notify the user via the notification framework in Android.

5.3. ContentProvider

A *content provider* provides a structured interface to application data. Via a *content provider* your application can 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 *content provider*.

broadcast receivers can be registered to receive system messages and *intents*. A *broadcast receiver* gets notified by the Android system, if the specified event occurs.

For example you can register a *broadcast receivers* for the event that the Android system completed the boot processor or for the event that the state of the phone changes, e.g. someone is calling.

5.5. (HomeScreen) Widgets

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

5.6. Live Wallpapers

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

6. Android Development Tools

6.1. Android SDK

The *Android Software Development Kit* (SDK) contains the necessary tools to create, compile and package Android application. Most of these tools are command line based.

The Android SDK also provides an Android device emulator, so that Android applications can be tested without a real Android phone. You can create *Android virtual devices* (AVD) via the Android SDK, which run in this emulator.

The Android SDK contains the *Android debug bridge* (adb) tool which allows to connect to an virtual or real Android device.

6.2. Android Development Tools

Google provides the *Android Development Tools* (ADT) to develop Android applications with Eclipse. ADT is a set of components (plug-ins) which extend the Eclipse IDE with Android development capabilities.

ADT contains all required functionalities to create, compile, debug and deploy Android applications from the Eclipse IDE. ADT also allows to create and start AVDs.

The Android Development Tools (ADT) provides specialized editors for resources files, e.g. layout files. These editors allow to switch between the XML representation of the file and a richer user interface via tabs on the bottom of the editor.

6.3. Dalvik Virtual Machine

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

Therefore you cannot directly run Java class files on Android, they need to get converted in the Dalvik bytecode format.

6.4. How to develop Android Applications

The Android SDK contains a tool called *dx* which converts Java class files into a *.dex* (Dalvik Executable) file. All class files of one application are placed in one compressed *.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 once 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 packaging.

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.

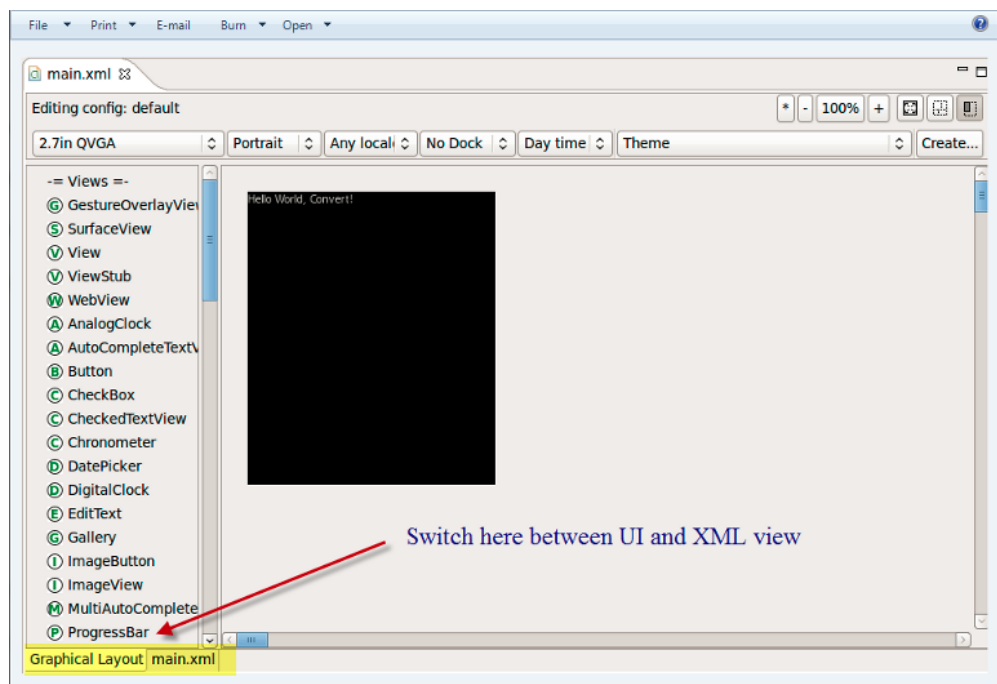
The Android Development Tools (ADT) performs these steps transparently to the user.

If you use the ADT tooling you press a button the whole Android application (*.apk* file) will be created and deployed.

6.5. Resource editors

The ADT allows the developer to define certain artifacts, e.g. Strings and layout files, in two ways: via a rich editor, and directly via XML. This is done via multi-page editors in Eclipse. In these editors you can switch between both representations by clicking on the tab on the lower part of the screen.

For example if you open the *res/layout/main.xml* file in the *Package Explorer* View of Eclipse, you can switch between the two representations as depicted in the following screenshot.



7. Android Application Architecture

7.1. AndroidManifest.xml

For example all *activities* and *services* of the application must be declared in this file.

It must also contain the required permissions for the application. For example if the application requires network access it must be specified here.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.vogella.android.temperature"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Convert"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="9" />
</manifest>
```

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. Therefore it is a good habit to use your reverse domain name as package name. 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 determine based on the *versionCode*, if it should perform an update of the applications for the existing installations. You typically start with "1" and increase this value by one, if you roll-out a new version of your application.

The *<activity>* tag defines an *activity*, in this example pointing to the *Convert* class in the *de.vogella.android.temperature* package. An intent filter is registered for this class which defines that this *activity* is started once the application starts (action *android:name="android.intent.action.MAIN"*). The category definition *category android:name="android.intent.category.LAUNCHER"* defines that this application is added to the application directory on the Android device.

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

The *uses-sdk* part of the *AndroidManifest.xml* file defines the minimal SDK version for which your application is valid. This will prevent your application being installed on unsupported devices.

7.2. Activities and Lifecycle

The Android system controls the lifecycle of your application. At any time the Android system may stop or destroy your application, e.g. because of an incoming call. The Android system defines a lifecycle for *activities* via predefined methods. The most important methods are:

- *onSaveInstanceState()* - called after the Activity is stopped. Used to save data so that the Activity can restore its states if re-started
- *onPause()* - always called if the Activity ends, can be used to release resource or save data

7.3. Configuration Change

An `Activity` will also be restarted, if a so called "configuration change" happens. A configuration change happens if an event is triggered which may be relevant for the application. For example if the user changes the orientation of the device (vertically or horizontally). Android assumes that an `Activity` might want to use different resources for these orientations and restarts the `Activity`.

In the emulator you can simulate the change of the orientation via **Ctrl+F11**.

You can avoid a restart of your application for certain configuration changes via the `configChanges` attribute on your `Activity` definition in your `AndroidManifest.xml`. The following `Activity` will not be restarted in case of orientation changes or position of the physical keyboard (hidden / visible).

```
<activity android:name=".ProgressTestActivity"
    android:label="@string/app_name"
    android:configChanges="orientation|keyboardHidden|keyboard">
</activity>
```

7.4. Context

The class `android.content.Context` provides the connection to the Android system and the resources of the project. It is the interface to global information about the application environment.

The *Context* also provides access to Android *services*, e.g. the Location Service.

activities and *services* extend the `Context` class.

8. Resources

8.1. Support in Android for resource files

Android supports that resources, like images and certain XML configuration files, can be keep separate from the source code.

These resources must be defined in the *res* directory in a special folder dependent on their purpose. You can also append additional qualifiers to the folder name to indicate that the related resources should be used for special configurations, e.g. you can specify that a resource is only valid for a certain screen size.

The following table give an overview of the supported resources and their standard folder prefix.

Table 1. Resources

Resource	Folder	Description
Simple Values	/res/values	Used to define strings, colors, dimensions, styles and static arrays of strings or integers. By convention each type is stored in a separate file, e.g. strings are defined in the <i>res/values/strings.xml</i> file.
Layouts	/res/values	XML file with layout description files used to define the user interface for <i>activities</i> and <i>Fragments</i> .
Styles and Themes	/res/values	Files which define the appearance of your Android application.
Animations	/res/animator	Define animations in XML for the property animation API which allows to animate arbitrary properties of objects over time.

The *gen* directory in an Android project contains generated values. *R.java* is a generated class which contains references to certain resources of the project.

If you create a new resource, the corresponding reference is automatically created in *R.java* via the Eclipse ADT tools. These references are static integer values and define IDs for the resources.

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

For example to access a String with the `R.string.yourString` ID, you would use the `getString(R.string.yourString)` method.

R.java is automatically created by the Eclipse development environment, manual changes are not necessary and will be overridden by the tooling.

8.2. Defining IDs

Android allows that you define ID of user interface components dynamically in the layout files, via the `@id/your_id` notation.

To control your IDs you can also create a file called *ids.xml* in your */res/values* folder and define all IDs in this file.

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

    <item name="button1" type="id"/>

</resources>
```

This allow you to use the ID directly 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>
```

9. Using Resources

9.1. Reference to resources in code

The `Resources` class allows to access individual resources. An instance of `Resources` can get access via the `getResources()` method of the `Context` class.

The `Resources` class is also used by other Android classes, for example the following code shows

```
BitmapFactory.decodeResource(getResources(), R.drawable.ic_action_search);
```

9.2. 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 a XML resource, you can refer to it via `@color/your_id`. Or if you defined a "hello" string in an XML resource, you could access it via `@string/hello`.

9.3. Activities and Layouts

The user interface for *activities* is defined via layouts. The layout defines the included *views* (widgets) and their properties.

A layout can be defined via Java code or via XML. In most cases the layout is defined as an XML file.

XML based layouts are defined via a resource file in the `/res/layout` folder. This file specifies the *ViewGroups*, *Views*, their relationship and their attributes for this specific layout.

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 `@+id/yourvalue`. The following shows an example in which a `Button` gets the `button1` ID assigned.

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

By conversion this will create and assign a new *yourvalue* ID to the corresponding *View*. In your Java code you can later access a *View* via the method `findViewById(R.id.yourvalue)`.

Defining layouts via XML is usually the preferred way as this separates the programming logic from the layout definition. It also allows the definition of different layouts for different devices. You can also mix both approaches.

10. Assets

10.1. 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.

10.2. Accessing assets

You access this data via the *AssetsManager* which you can access the `getAssets()` method.

The *AssetsManager* class allows 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 a Bitmap from Assets
```

```

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();
        }
    }
}
}

```

11. Installation

11.1. Options

You have different options to install the Android development tools. The simplest way is to download a full packaged pre-configured Eclipse.

For other options please see [Android installation](#)

11.2. Standalone ADT installation

11.2.1. Download

Google provides a pre-packaged and configured Eclipse based Android development environment. The following link allows to download a archive file which includes all required tools for Android development.

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

11.2.2. Standalone ADT installation

Extract the zip file and start Eclipse from the *eclipse* folder via the *eclipse* native launcher, e.g. *eclipse.exe* under Windows.

12. Android virtual device - Emulator

12.1. What is the Android Emulator?

The Android Development Tools (ADT) include an emulator to run an Android system. The emulator behaves like a real Android device (in most cases) and allows you to test your application without having a real device.

You can configure the version of the Android system you would like to run, the size of the SD card, the screen resolution and other relevant settings. You can define several of them with different configurations.

These devices are called *Android Virtual Device* and you can start several in parallel.

12.2. Google vs. Android AVD

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

created for the Google API's will also contain several Google applications, most notable the Google Maps application.

If you want to use functionality which is only provided via the Google API's, e.g. Google Maps you must run this application on an AVD with Google API's.

12.3. Emulator Shortcuts

The following shortcuts are useful for working with the emulator.

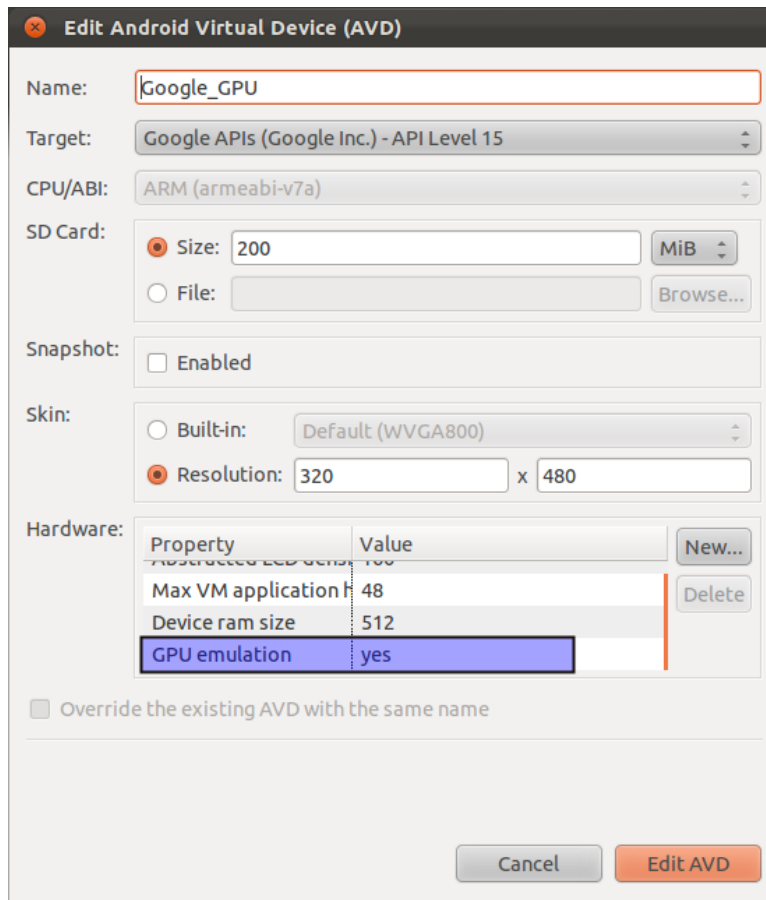
Alt+Enter Maximizes the emulator. Nice for demos.

Ctrl+F11 changes the orientation of the emulator.

F8 Turns network on / off.

12.4. Parameter

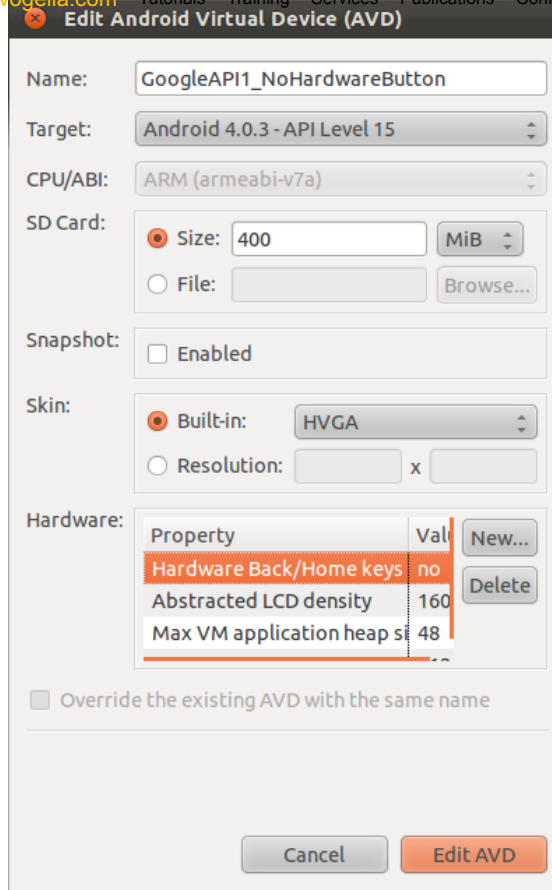
The graphics of the emulator can use the native GPU of the computer. This makes the rendering in the emulator very fast. To enable this, add the `GPU Emulation` property to the device configuration and set it to `true`.



Property	Value
Max VM application heap size	48
Device ram size	512
GPU emulation	yes

You can also set the `Enabled` flag for Snapshots. This will save the state of the emulator and will let it start much faster. Unfortunately currently native GPU rendering and Snapshots do not work together.

Android devices do not have to have hardware button. If you want to create such an AVD, add the `Hardware Back/Home keys` property to the device configuration and set it to `false`.



Edit Android Virtual Device (AVD)

Name:

Target:

CPU/ABI:

SD Card:

☒ Size:

☐ File:

Snapshot: ☐ Enabled

Skin:

☒ Built-in:

☐ Resolution:

Hardware:

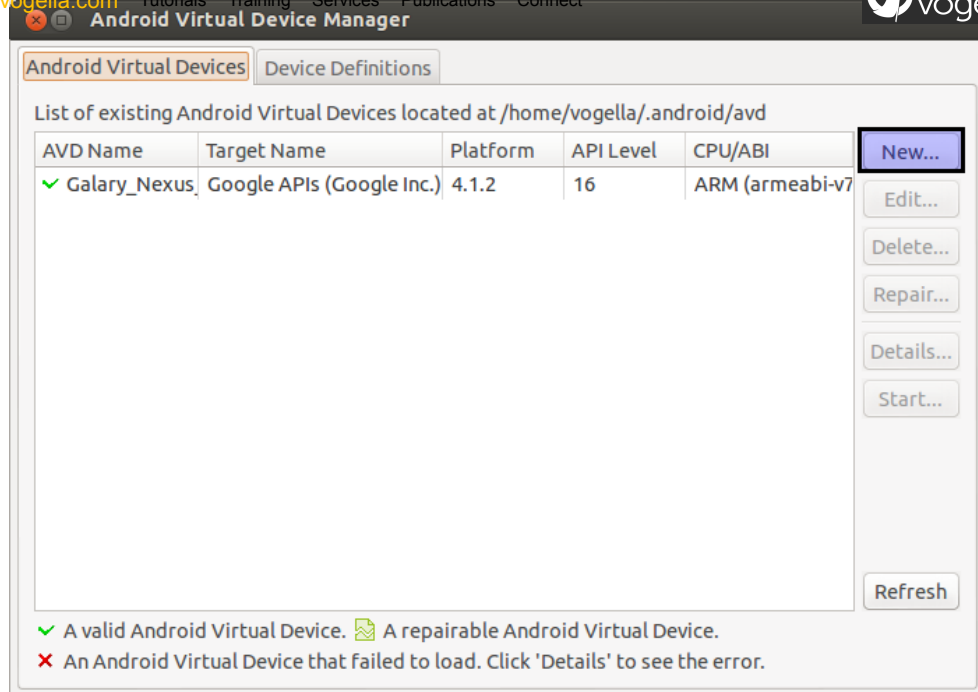
Property	Value
Hardware Back/Home keys	no
Abstracted LCD density	160
Max VM application heap size	48

☐ Override the existing AVD with the same name

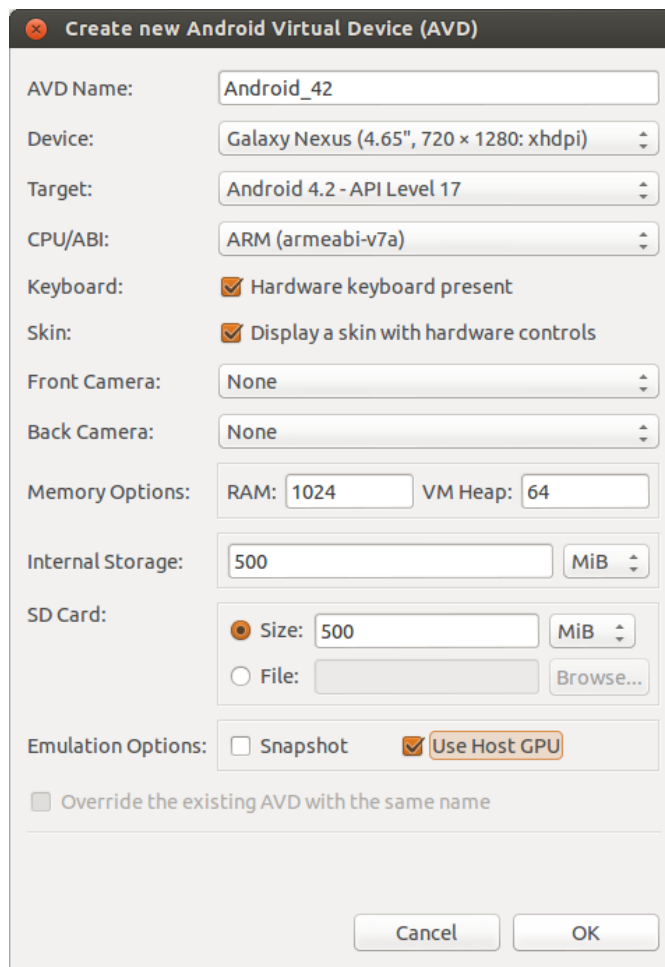
13. Tutorial: Create and run Android Virtual Device

13.1. Create AVD

To define an Android Virtual Device (ADV) open the *AVD Manager* dialog via *Window* → *Android Virtual Device Manager* and press the *New* button.



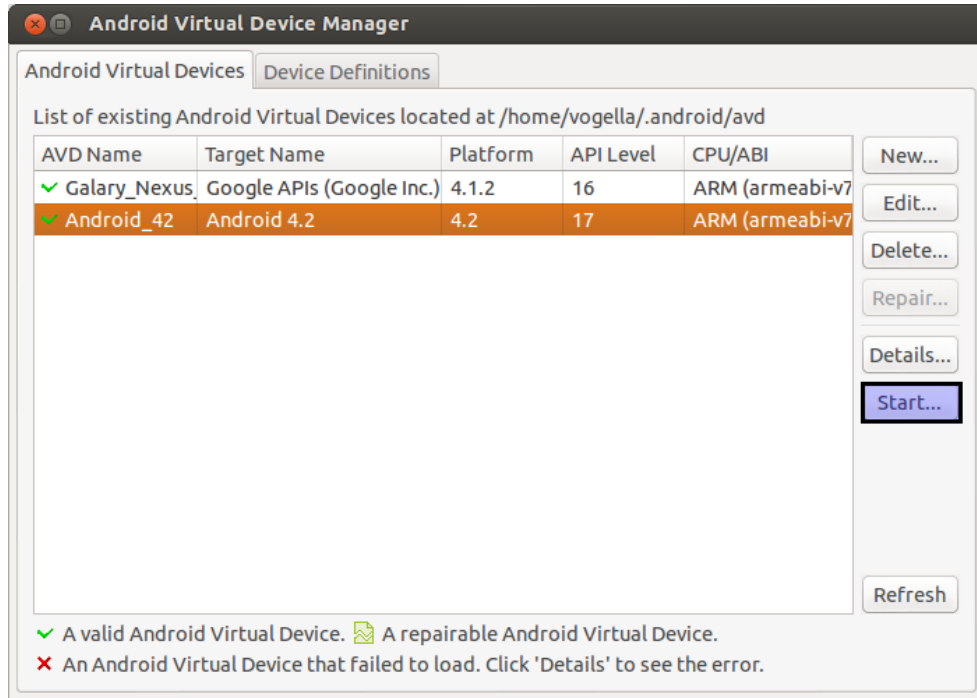
Enter the values similar to the following screenshot.



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

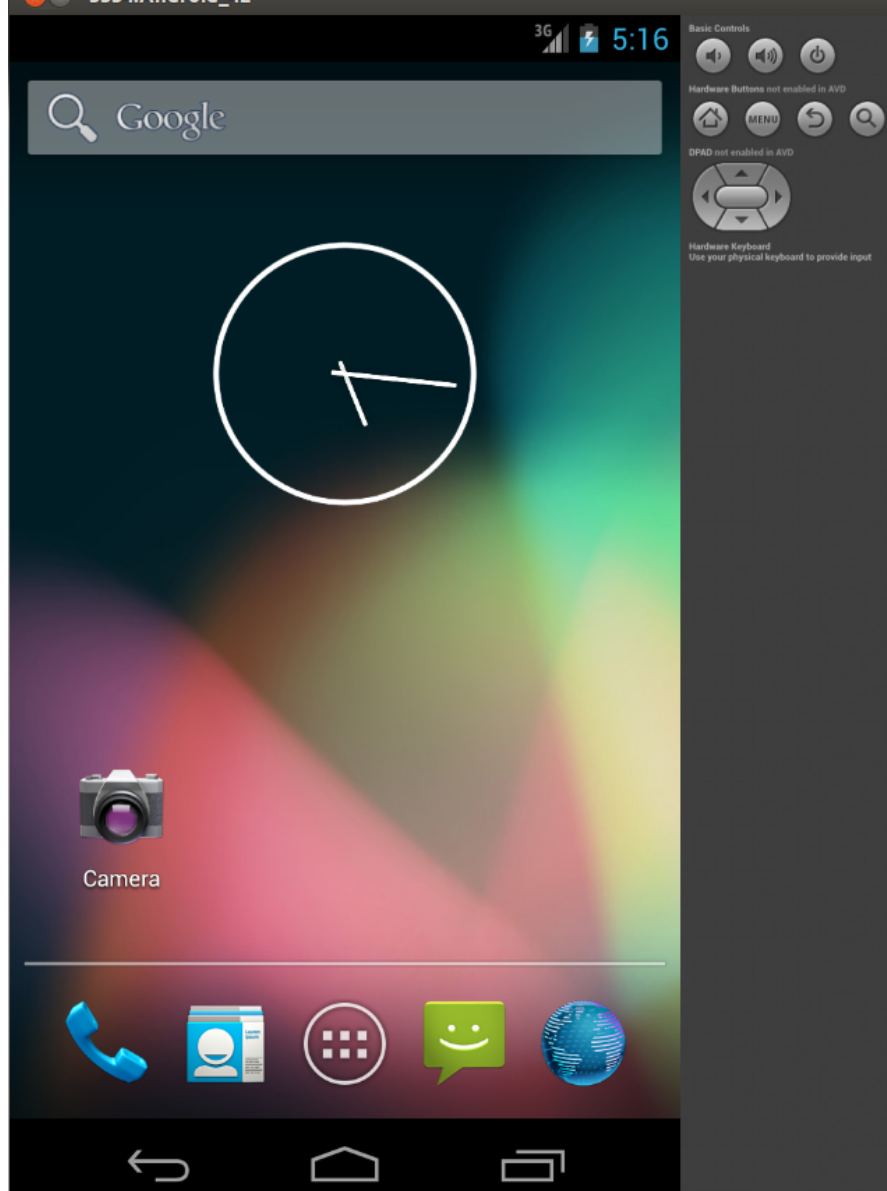
13.2. Run AVD

To test if your setup is correct, select your your new entry and press the *Start* button



After some time your AVD starts. Do not interrupt this startup process, as this might corrupt the AVD.

After the AVD started, you can use the AVD via the mouse and via the virtual keyboard of the emulator.



13.3. Stopping the emulator

During development you don't stop the AVD, you just re-deploy your application.

14. 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](#).

15. Conventions and API level

15.1. API version

The tutorials of this document have been developed and tested with Android 4.2, API Level 17. Please use this version for all tutorials in this tutorial. Higher versions of the API level should also work. A

15.2. Android project and package name

The base package for the projects is always the same as the project name, e.g. if you are asked to create a project called *de.vogella.android.example.test*, then the corresponding package name is *de.vogella.android.example.test*.

The application name, which must be entered on the Android project generation wizard, will not be predefined. Choose a name you like.

16. Tutorial: create and run Android project

In this section you create a simple Android project and run it.

You create an Android application with the data from the following table. The process of creating the Android application is described and depicted below the table.

Table 2. New Android project

Property	Value
Application Name	Test App
Project Name	com.vogella.android.first
Package name	com.vogella.android.first
Template	BlankActivity
Activity	MainActivity
Layout	activity_main

To create a new Android project select *File* → *New* → *Other* → *Android* → *Android Project* from the menu. Enter the fitting data from the table above in the first wizard page.

New Android Application
Creates a new Android Application

Application Name:

Project Name:


Package Name:


Minimum Required SDK:

Target SDK:


Compile With:

Theme:


 The application name is shown in the Play Store, as well as in the Manage Application list in Settings.



Press the *Next* button and ensure that you have selected to create a launcher icon and an *activity*.



New Android Application



Configure Project

☒ Create custom launcher icon

☒ Create activity

☐ Mark this project as a library

☒ Create Project in Workspace


Location:


Working sets

☐ Add project to working sets

 Working sets:

On the wizard page for the launcher icon, create a nice looking icon. The following screenshot shows an example.



New Android Application



Configure Launcher Icon

Configure the attributes of the icon set

Foreground: Image Clipart Text

Choose... 

☒ Trim Surrounding Blank Space

Additional Padding:


Foreground Scaling: Crop Center


Shape None Square Circle


Background Color:


Foreground Color:

Preview:

ldpi: 

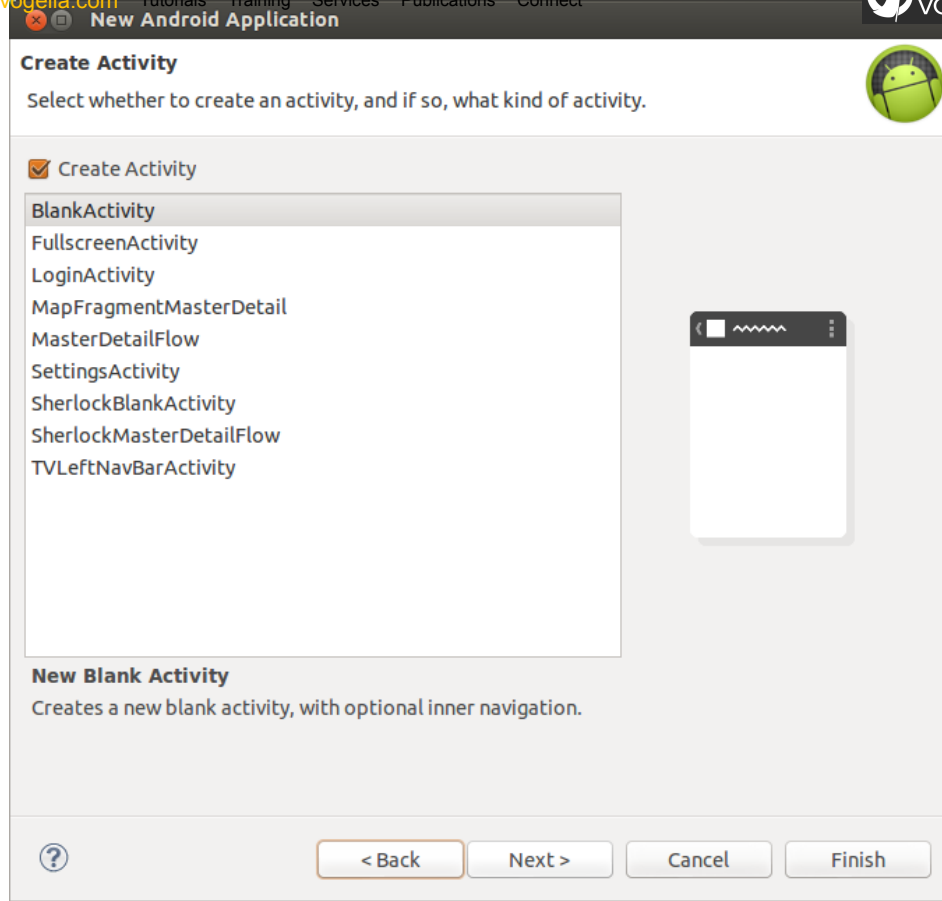
mdpi: 

hdpi: 

xhdpi: 

?
< Back
Next >
Cancel
Finish

Press the *Next* button and select on the next page the *BlankActivity* template. Press the *Next* button



Enter the following data which was also described in the above table.

New Android Application

New Blank Activity

Creates a new blank activity, with optional inner navigation.

Activity Name

Layout Name

Navigation Type

The name of the activity class to create

?

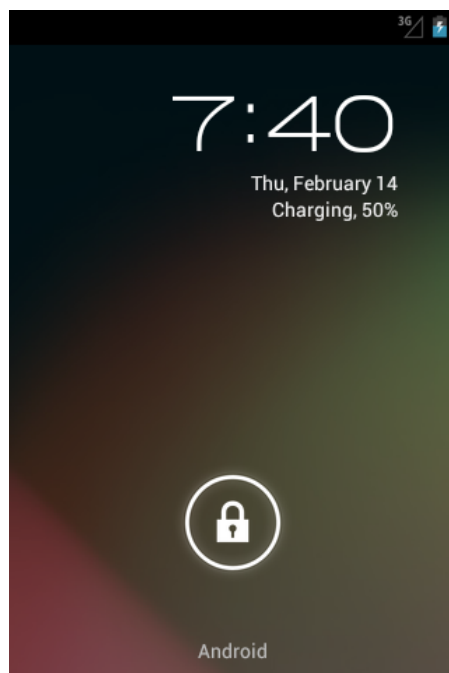
< Back

Next >

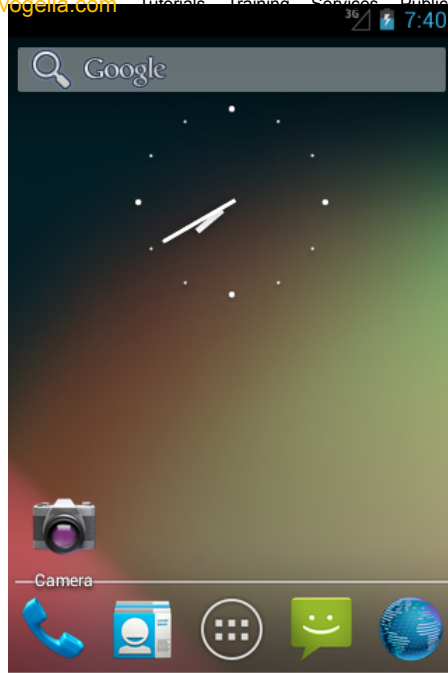
Cancel

Finish

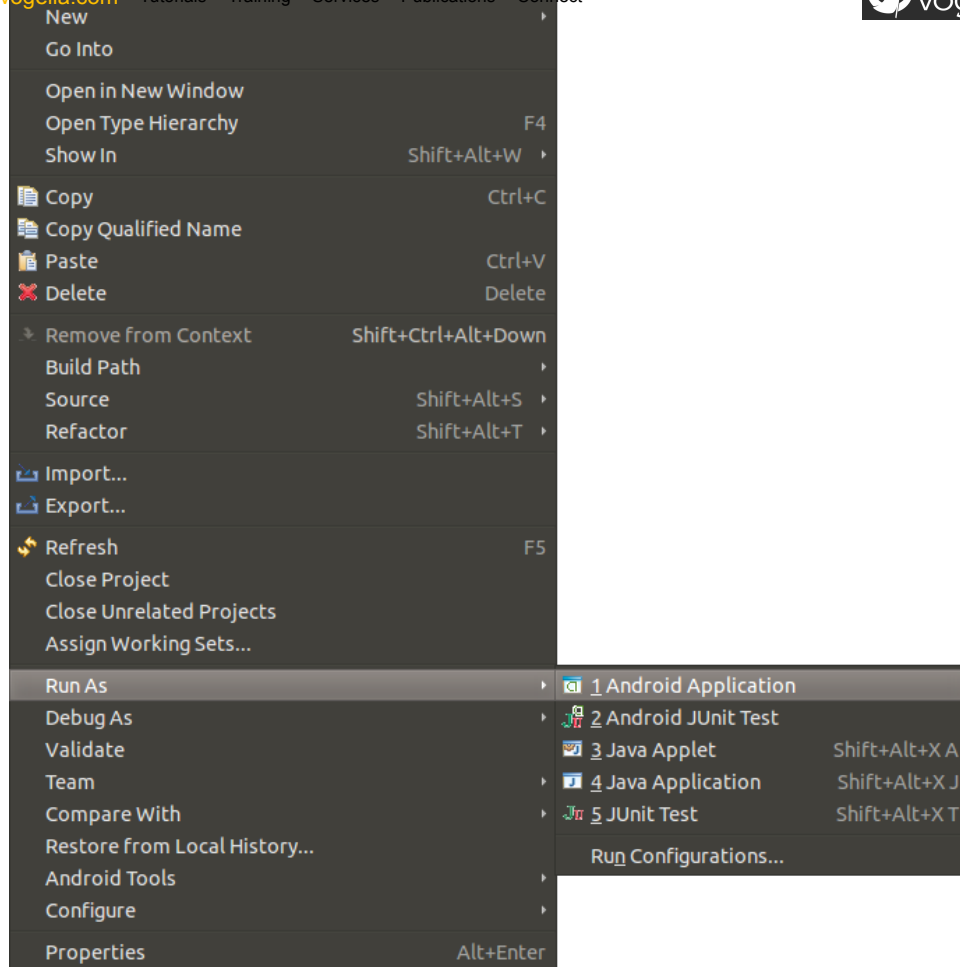
If you have not yet done so, create an Android virtual device (AVD) fitting for your selected API version and start this AVD. Wait until the AVD has started.



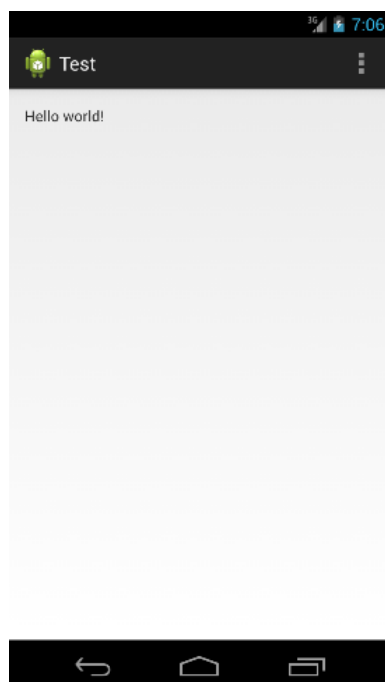
Unlock your emulator.



Start your Android application on the emulator. To build, install and run your application the Android Application, select your project, right click on it, and select *Run-As → Android Application*.



This starts your application on the AVD. The started application is a simple *Hello, world.* application.



17. Views

17.1. Available widgets in Android

Android provides lots of simple *views* (widgets), e.g. the `Button`, `TextView`, `EditText` classes and well as more complex widgets, for example `ListView` or `GridView` to show structured data.

17.2. View class

All *views* in Android extends the `android.view.View` class. This class is relatively larger (greater than 18 000 lines of code) and provides a lot of base functionality for subclasses. Customer can implement their own views by extending `android.view.View`.

18. Tutorial: Create a temperature converter

18.1. Install the demo application

This application is available on the Android Marketplace under [Android Temperature converter](#).

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



18.2. Create Project

Select *File* → *New* → *Other* → *Android* → *Android Application Project* to create a new Android project.

Use *Temperature Converter* as *Application name* and *de.vogella.android.temperature* as project and package name. Select the latest Android SDK for *Minimum SDK*, *Target SDK* and *Compile with target*. After entering this data, press the *Next* button.

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.



Leave the default settings on the next wizard page and click the *Next* button.



New Android Application



Configure Project

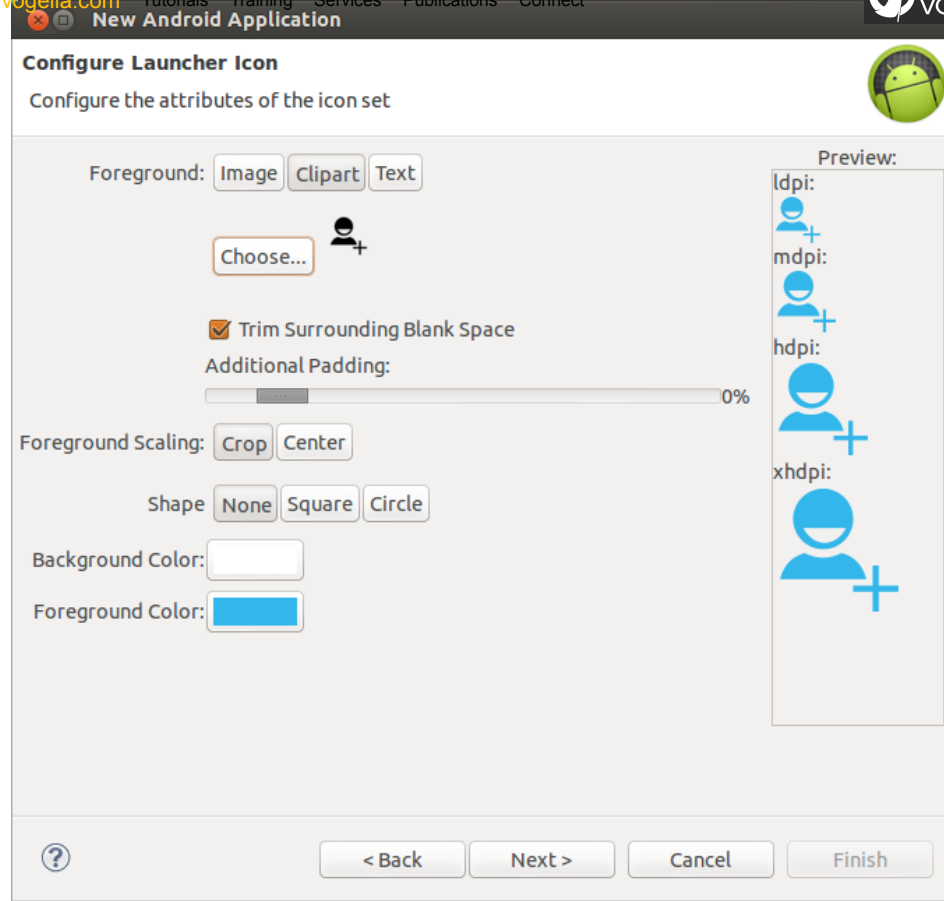
☒ Create custom launcher icon
☒ Create activity
☐ Mark this project as a library
☒ Create Project in Workspace

Location:

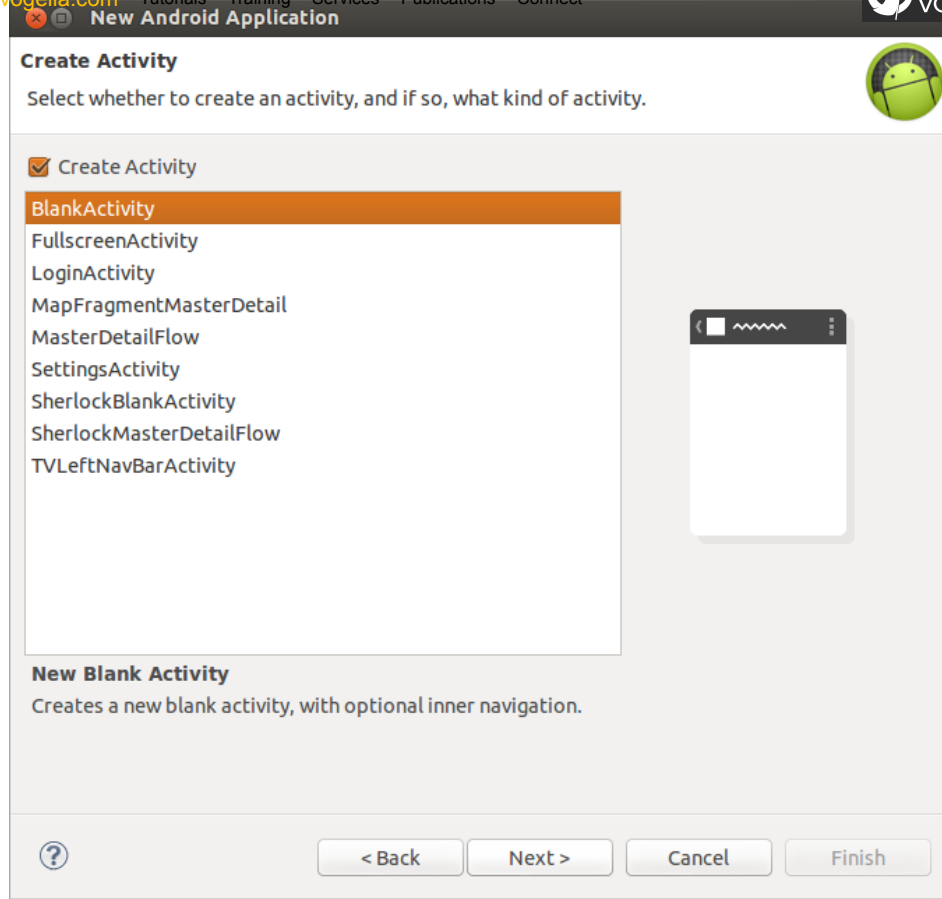
Working sets

☐ Add project to working sets
 Working sets:

The next screen allows you to create a *launcher icon* for your application. Modify the icon to your liking and press the *Next* button.



Select the *BlankActivity* template and press the *Next* button.



On the next dialog ensure that the name of the Activity is set to *MainActivity* and the layout name is set to *activity_main*.


New Android Application

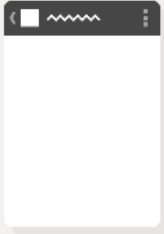
New Blank Activity
Creates a new blank activity, with optional inner navigation.


Activity Name


Layout Name

Navigation Type

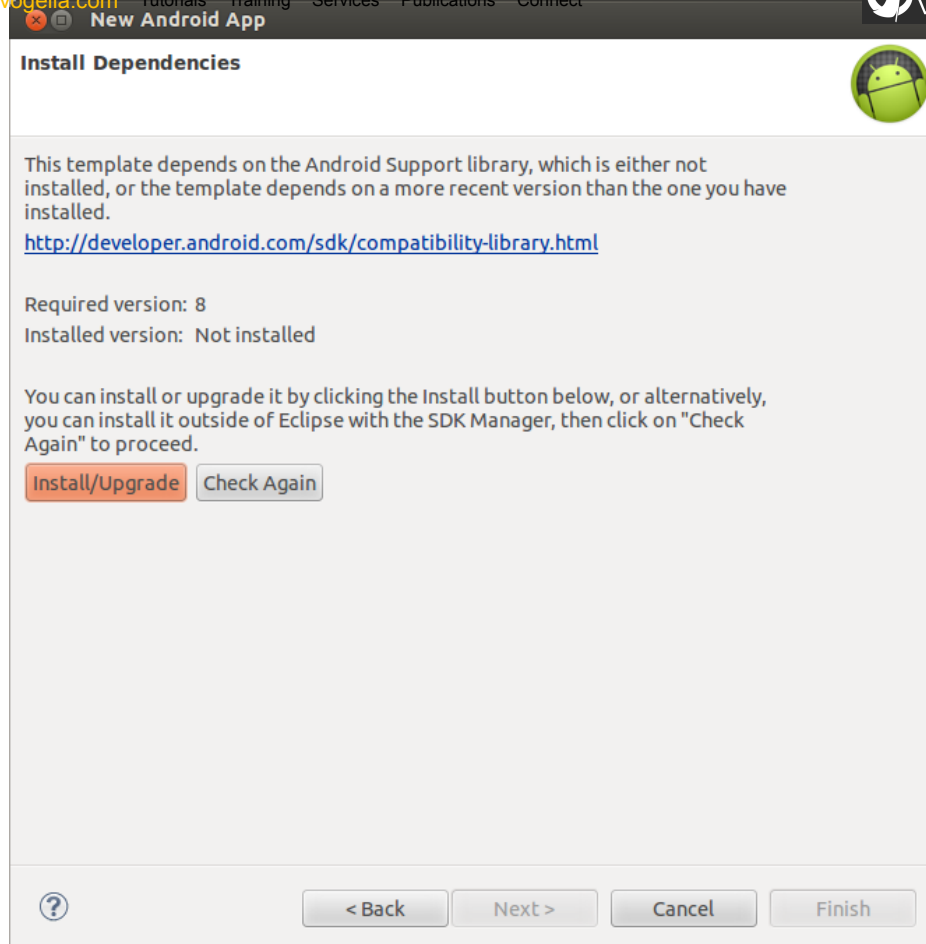




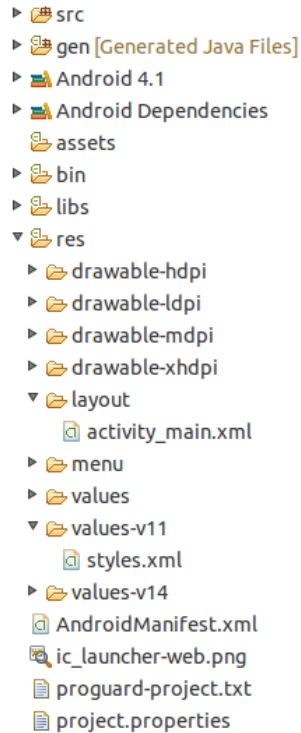
 The name of the activity class to create



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



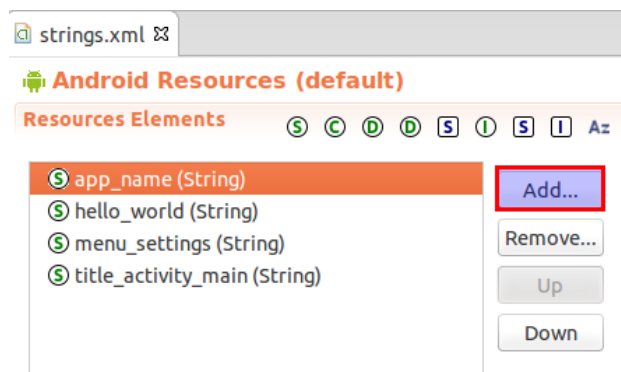
After the wizard ends, a project structure similar to the following picture is created.



18.3. Create attributes

Android allows you to create static attributes, e.g. Strings or colors. These attributes can for example be used in your XML layout files or referred to via Java source code.

Select the `res/values/string.xml` file and press the *Add* button.



Select the *Color* entry in the following dialog and press the *OK* button. Enter `myColor` as the name and `#F5F5F5` as the value.

© A [color](#) value specifies an RGB value, can be used in various places such as Drawable or the color to use for text character and then is followed by the information in one of the following formats: #RRGGBB or #AARRGGBB.

Name

Value*

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

Table 3. String Attributes

Name	Value
celsius	to Celsius
fahrenheit	to Fahrenheit
calc	Calculate

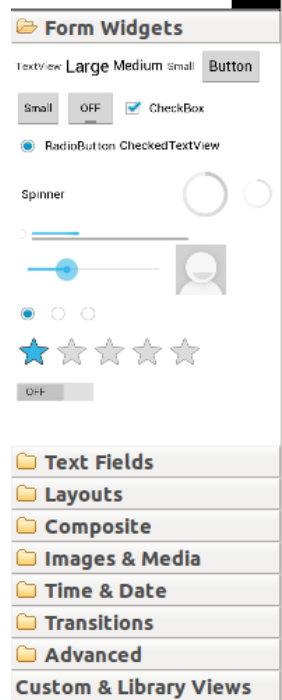
Switch to the XML representation and validate that the values are correct.

```
<resources>
    <string name="app_name">Temparature Convertor</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">Temparature Convertor</string>
    <color name="myColor">#3399CC</color>
    <string name="celsius">to Celsius</string>
    <string name="fahrenheit">to Fahrenheit</string>
    <string name="calc">Calculate</string>
</resources>
```

18.4. Add Views

Select the `res/layout/activity_main.xml` file and open the Android editor via a double-click. 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*.

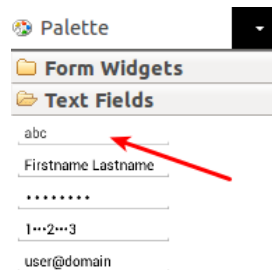
The following shows a screenshot of the *Palette* side of this editor, from which you can drag and drop new user interface components into your layout. Please note that the *Palette* view changes frequently so your view might be a bit different.



You will now create the layout for your Android 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).



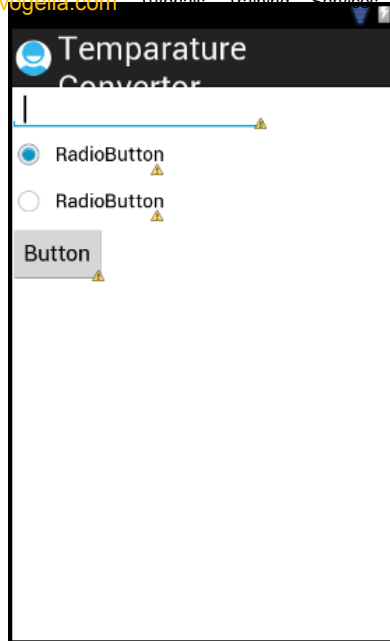
All entries in the *Text Fields* section define text fields. The different entries define additional attribute for them, e.g. if a text field should only contain numbers.

Drag this onto the layout to create a text input field.

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 very fast, 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" >

    <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" >

        <requestFocus />
    </EditText>

    <RadioGroup
        android:id="@+id/radioGroup1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        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_alignParentLeft="true"
        android:layout_below="@+id/radioGroup1"
        android:text="Button" />
</RelativeLayout>
```

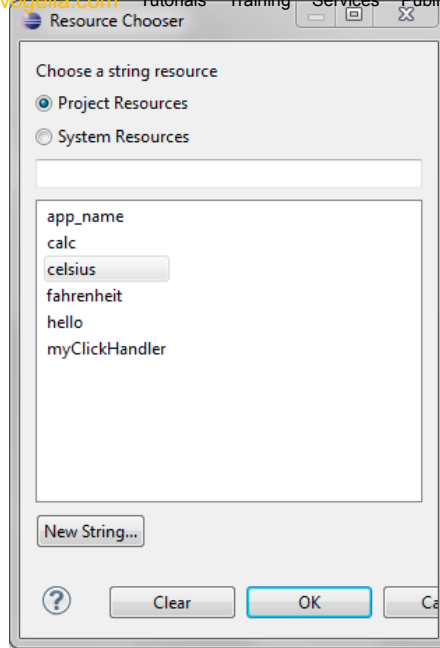
18.5. Edit view properties

If you select a user interface component (an instance of `View`), you can change its properties via the Eclipse *Properties view*. Most of the properties can be changed via the menu which can be opened via right-click. You can also edit properties of fields directly in XML. Changing properties in the XML file is much faster, if you know what you want to change. But the right-click menu is nice, if you are searching for a certain property.

Open your layout file.

Use a right-click on the first radio button to assign the `celsius` String attribute to its `text` property. Assign the `fahrenheit` string attribute to the `text` property of the second radio button.

Edit Text...	
Edit ID...	
Edit Style...	
Layout Width	▸
Layout Height	▸
Other Properties	▸
Extract Include...	
Extract Style...	
Wrap in Container...	
Remove Container...	
Change Widget Type...	
radioGroup1	▸
RelativeLayout	▸
Select	▸
Cu <u>t</u>	Ctrl+X
<u>C</u> opy	Ctrl+C
Paste	Ctrl+V
<u>D</u> ele <u>t</u> e	Delete
Play Animation	▸
Export Screenshot...	
Show Included In	▸
Sho <u>w</u> In	▸
Input Methods	▸



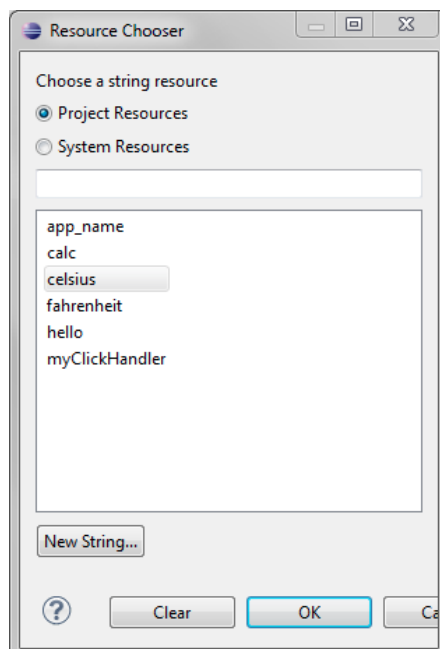
From now on, I assume you are able to use the properties menu on user interface components. You can always either edit the XML file or modify the properties via right-click.

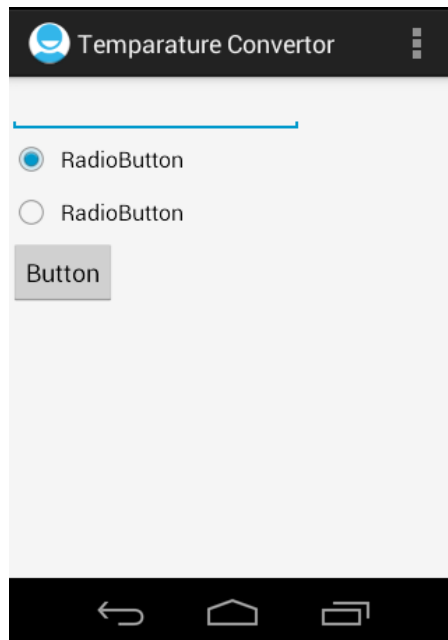
Set the *checked* property to true for the first RadioButton.

Assign *calc* to the text property of your button and assign the value *onClick* to the *onClick* property.

Set the *Input type* property to *numberSigned* and *numberDecimal* on the *EditText*.

All your user interface components are contained in a layout. Assign a background color to this *Layout*. Right-click on an empty space in *Graphical Layout* mode, then select *Other Properties* → *All by Name* → *Background*. Select *Color* and then select *myColor* in the dialog.





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: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"
        >

        <requestFocus />
    </EditText>

    <RadioGroup
        android:id="@+id/radioGroup1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        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_alignParentLeft="true"
        android:layout_below="@+id/radioGroup1"
        android:onClick="onClick"
        android:text="@string/calc" />

</RelativeLayout>

```

18.6. Change the Activity source code

During the generation of your new Android project you specified that an Activity called MainActivity should be created. The project wizard created the corresponding Java class.

Change your MainActivity class to the following listing. Note that the onClick will be called based on the onClick property of your button. I use the same name as this is easier to remember.

```

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

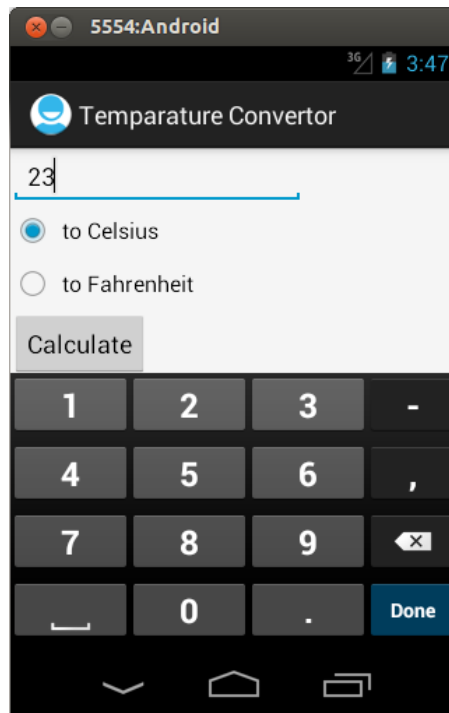
        // Converts to celsius
        private float convertFahrenheitToCelsius(float fahrenheit) {
            return ((fahrenheit - 32) * 5 / 9);
        }

        // Converts to fahrenheit
        private float convertCelsiusToFahrenheit(float celsius) {
            return ((celsius * 9) / 5) + 32;
        }
    }
}

```

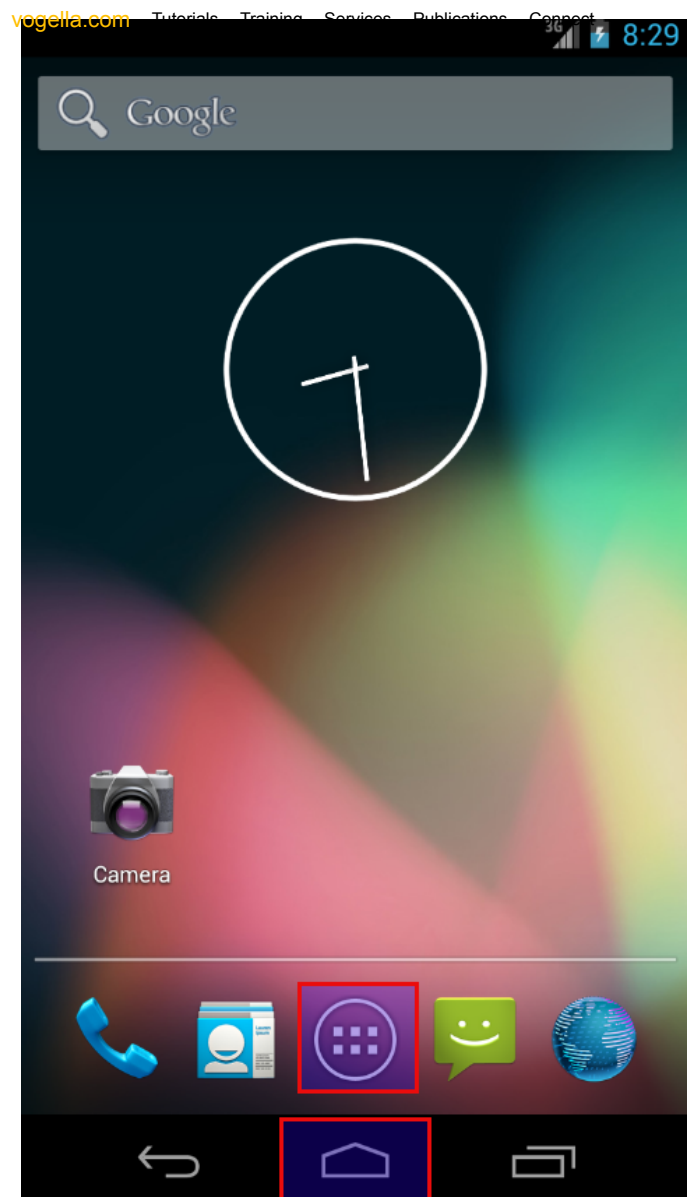
To start the Android Application, select your project, right click on it, and select *Run-As* → *Android Application*. If an emulator is not yet running, it will be started. Be patient, the emulator starts up very slowly.

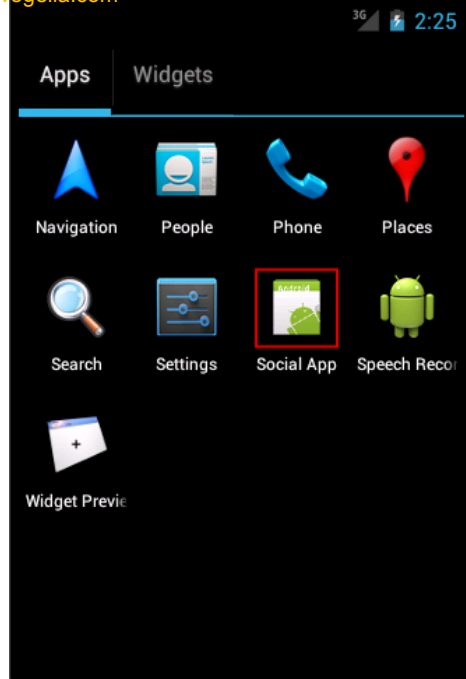
Type in a number, select your conversion and press the button. The result should be displayed and the other option should get selected.



19. Starting an installed application

After you run your application on the virtual device, you can start it again on the device. If you press the *Home* button you can select your application.





20. Layout Manager and ViewGroups

20.1. Available 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.

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

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

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

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

Table 4. 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.

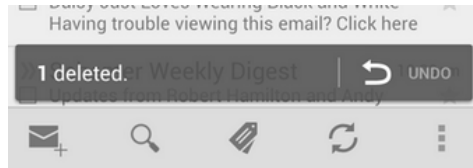
Widgets can use fixed sizes, e.g. with the `dp` definition, for example `100dp`. While `dp` is a fixed size it

The `match_parent` value tells the to maximize the widget in its parent. The `wrap_content` value tells the layout to allocate the minimum amount so that widget is rendered correctly.

20.2. FrameLayout

`FrameLayout` is a layout manager which draws all child elements on top of each other. Which 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.



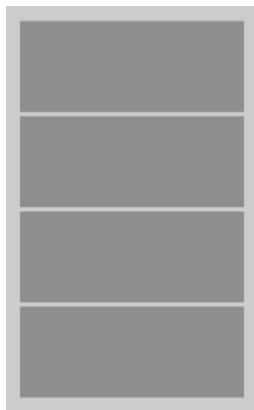
20.3. LinearLayout

`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`, `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.

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 have always a certain ratio.

20.4. RelativeLayout

`RelativeLayout` allow to position the widget relative to each other. This allows 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>
```

20.5. 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 for define for each view in which row and column it should be placed and 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: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>

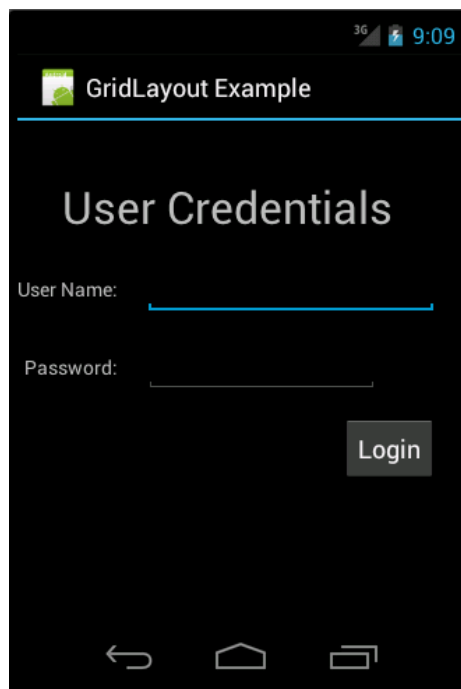
        <EditText
            android:id="@+id/input1"
            android:layout_column="1"
            android:layout_columnSpan="2"
            android:layout_row="2"
            android:ems="8" />

        <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.



20.6. ScrollView

The `ScrollView` class can be used to contain one view that might be too big to fit on one screen. `ScrollView` will in this case 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: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.



21. Tutorial: ScrollView

Create an android project "de.vogella.android.scrollview" with the activity "ScrollView". Create the following layout and class.

```

<?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
    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>

```

```

package de.vogella.android.scrollview;

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

public class ScrollView extends Activity {

    /** Called when the activity is first created. */

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView view = (TextView) findViewById(R.id.TextView02);
        String s="";
        for (int i=0; i < 100; i++) {
            s += "vogella.com ";
        }
        view.setText(s);
    }
}

```



22. DDMS perspective and important views

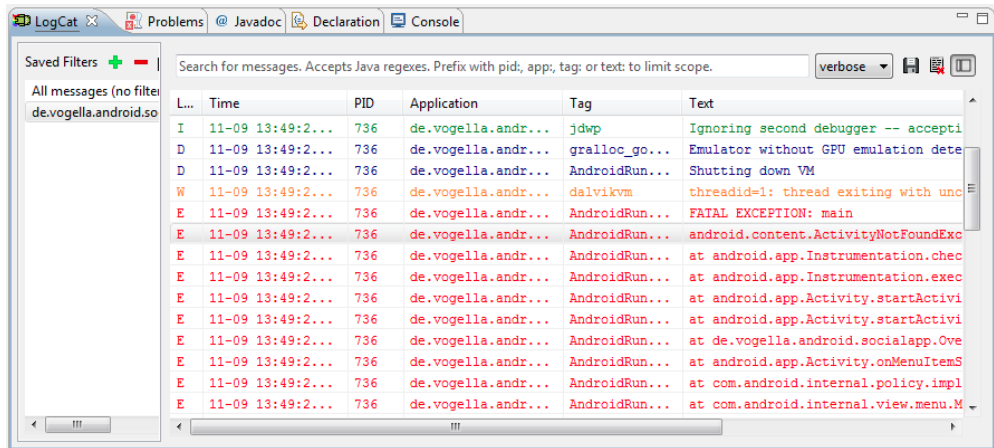
22.1. DDMS - Dalvik Debug Monitor Server

application program. Select **Window** → **Open Perspective** → **Other** → **DDMS** to open this perspective. It includes several views which can also be used independently and allows for example the application to place calls and send SMS to the device. It also allows the application to set the current geo position and allows you to perform a performance trace of your application.

22.2. LogCat View

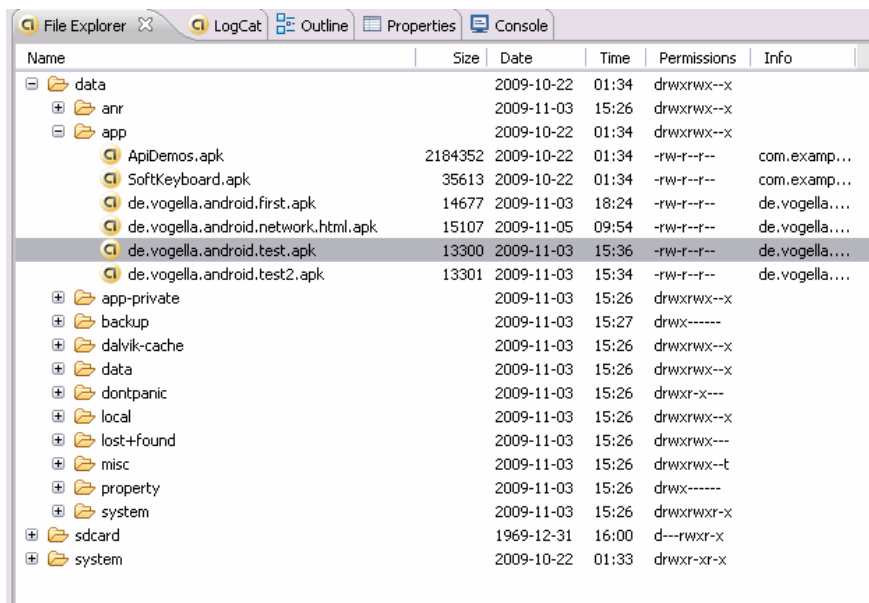
You can see the Android log statements via the *LogCat view*.

You can open this view via **Window** → **Show View** → **Other** → **Android** → **LogCat**.



22.3. File explorer

The file explorer allows to see the files on the Android simulator.



23. Deployment

23.1. Overview

USB, email yourself the application or use one of the many Android markets to install the application.

The following describes the most common ones.

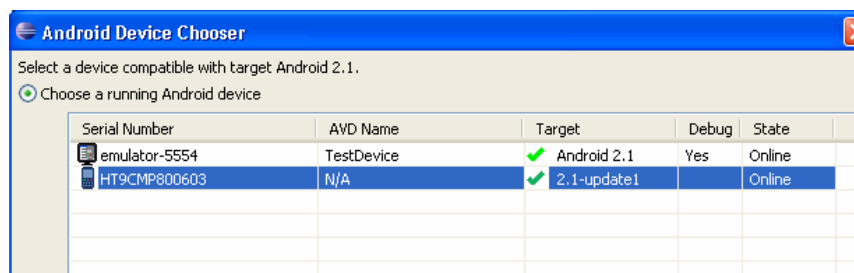
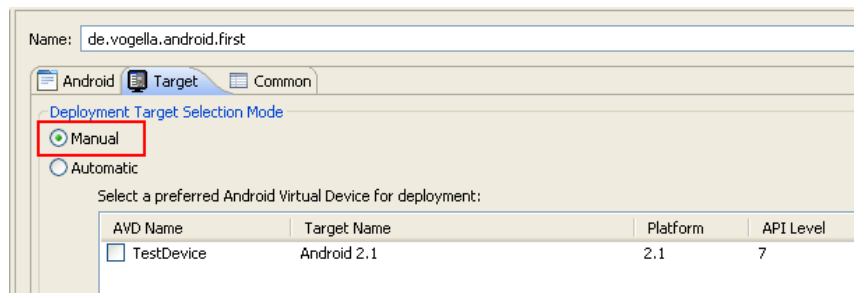
23.2. Deployment via Eclipse

Turn on *USB Debugging* on your device in the settings. Select in the settings of your device *Applications* → *Development*, then enable USB debugging.

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

For details please see [Developing on a Device](#). Please note that the Android version you are developing for must be the installed version on your phone.

If you have only one device connected and no emulator running, the Android development tools will automatically deploy to this device. If you have several connected you can selected which one should be used.



23.3. Export your application

Android application 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 loose the key you will NOT be able to update your application ever again.

Make sure to backup your key.

23.4. Via external sources

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.

23.5. 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.

24. Thank you

Please help me to support this article:



25. Questions and Discussion

Before posting questions, please see the [vogella FAQ](#). If you have questions or find an error in this article please use the www.vogella.com [Google Group](#). I have created a short list [how to create good questions](#) which might also help you.

26. Links and Literature

26.1. Source Code

[Source Code of Examples](#)

26.2. Android Resources

[Android ListView and ListActivity](#)

[Android SQLite Database](#)

[Android Widgets](#)

[Android Live Wallpaper](#)

[Android Services](#)

[Android Location API and Google Maps](#)

[Android Intents](#)

[Android and Networking](#)

[Android Homepage](#)

[Android Developer Homepage](#)

[Android Google Groups](#)

[Android Live Folder](#)

26.3. vogella Resources

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

[Android Tutorial](#) Introduction to Android Programming

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

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

[JUnit Tutorial](#) Test your application

[Git Tutorial](#) Put everything you have under distributed version control system