

# Mobile Anwendungen mit Android

---

---

# Literatur

---

1. Ed Burnette: *Hello, Android*, Pragmatic Programmers, Third Edition, LLC 2010
  2. Wei-Meng Lee: *Android Application Development Cookbook*, Wrox 2013
  3. Satya Komatineni, Dave MacLean, Sayed Hashimi: *Pro Android 4*, apress 2013
  4. Dave Smith, Jeff Friesen: *Android Receipes*, 3. Edition apress 2013
  5. <http://developer.android.com>
  6. Reto Meier: *Professional Android 4 Development*, Wrox 2013
  7. Murat Aydin: *Android 4: New Features for Application Development*, Packt Publishing 2012
  8. Jim Wilson: *Creating Dynamic UI with Android Fragments*, Packt Publishing 2013
-

# Lernziele

---

- Programmierkenntnisse vertiefen.
- Softwareschnittstellen und Architektur von Android kennenlernen und verstehen.
- Selbstständige Einarbeitung in verschiedene Themen.
- Programmierung einfacher Android-Apps.

---

# Einführung

# Android Manifest

---

- Eine Activity muss als Eintrittspunkt der Anwendung definiert werden.


```
<activity
    android:name=".ManifestDemoActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```



## Erlaubnisse (Permissions)


- Android-Anwendung hat in der Regel kein Erlaubnis, auf andere Ressourcen zuzugreifen. Benötigt die Anwendung dennoch Zugriffe darauf, muss der Benutzer bei der Installation zustimmen. Erbetene Erlaubnisse müssen angegeben werden (für die Konstanten siehe auch **android.Manifest.permission**)

```
<permission android:name="android.permission.CAMERA" />
<permission android:name="android.permission.READ_CONTACTS" />
<permission android:name="android.permission.WRITE_CONTACTS" />
```

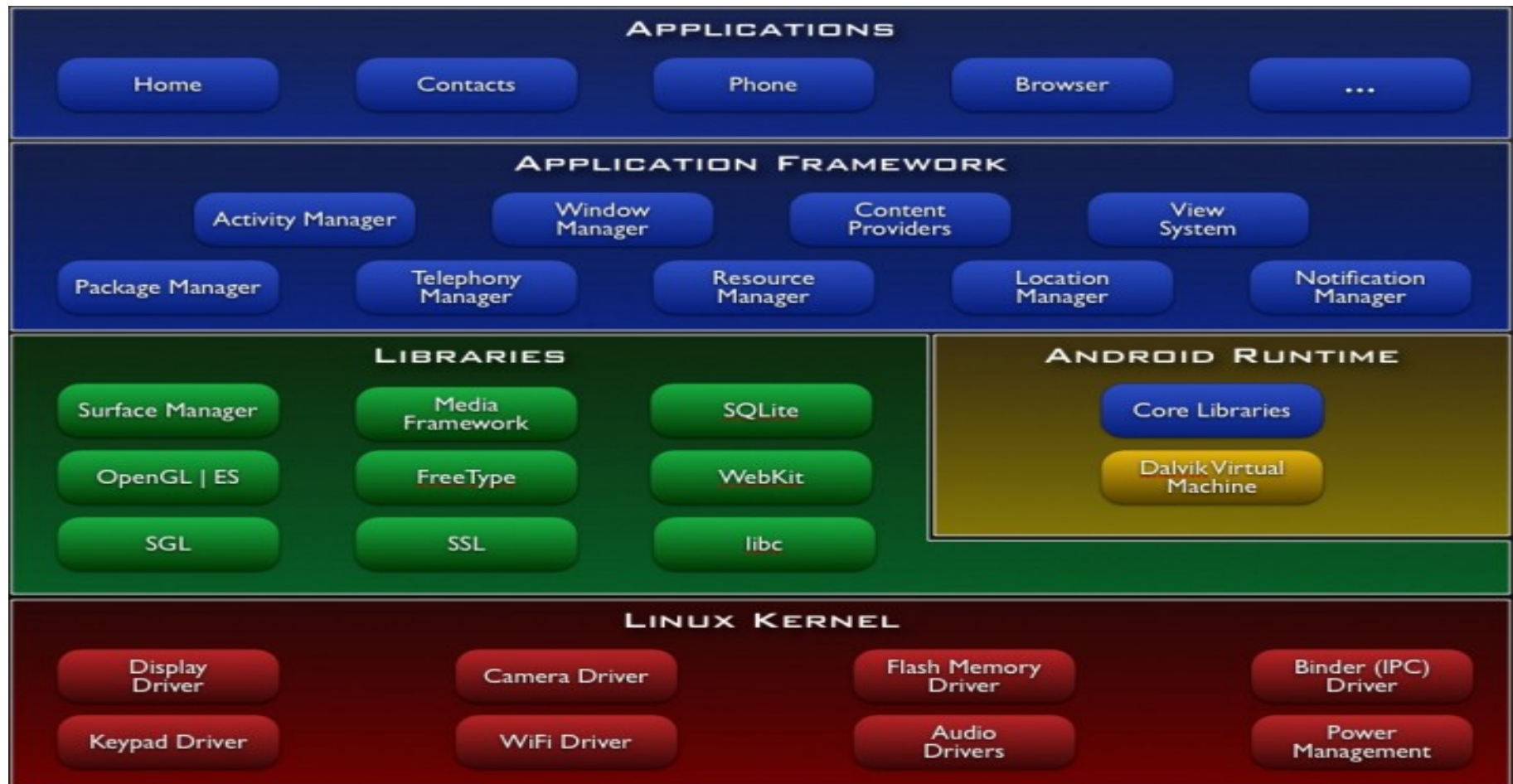


# Versionen

---

- 0.9 SDK-beta, Mitte August 2008
- weitere Versionen
- 2.3 (Gingerbread) 06.12.2010, Linux-Kernel 2.6.35.7, neuer Garbage Collector, Videoverbesserung, NFC ..
- 3.0 (Honeycomb) 22.02.2011, Linux-Kernel 2.6.36 - Hauptfokus Tablets
- 4.0 (Ice Cream Sandwich) 19.10.2011, Linux-Kernel 3.0.1.
- Weitere Versionen
- Aktuell Android ~~5.0.2~~ 

# Android-Architektur




Android-Architektur Quelle [5]

# Android-Architektur

---

## Applications


- Verschiedene fertige Kernanwendungen (Home, Browser, Telefon, Kontakte usw.).
- können von der eigenen Anwendung aus gestartet werden.
- Häufig in Java (ab Version 5) programmiert (auch C, C++ möglich).
- Android-SDK wird benötigt. 
- Java-Quelltext wird von einem Cross-Assembler für die Dalvik-VM angepasst.
- Alle Anwendungen sind generell gleichberechtigt und können jederzeit ausgetauscht werden.



# Android-Architektur

---

## Application Frameworks

- **Activity Manager** 
  - kontrolliert die Lebenszyklen der Apps
  - verwaltet das Navigieren zwischen Apps.
- **Content Providers** versorgen Apps mit gemeinsamen Daten (z.B. Photo- oder Musik-Sammlungen, Kontakte).
- **Location Manager** liefert die Position des Geräts.
- **Notification Manager**: informiert Benutzer über signifikante Ereignisse. 
- **Package Manager** verwaltet installierte Packages.

# Android-Architektur



---

- **Resource Manager**: Zur Verwaltung und Zugriffen von Ressourcen (wie String, Graphiken, XML-Dateien usw).
- **Telephony Manager** zuständig für Telefon-Services.
- **View System**: Komponente, die UI-Elemente verwaltet und Ereignisse generiert.
- **Window Manager**: Zur Verwaltung des Bildschirms.

# Android-Architektur

---


## Bibliotheken (Libraries)

- **FreeType**: Für Bitmap und Fonts. 
- **(Bionic)libc**: Angepasste und optimierte BSD-Implementierung. 
- **LibWebCore**: Basiert auf WebKit (Web Browser Engine, wird auch in Google Chrome und Apple Safari verwendet).
- **Media Framework**: Basiert auf OpenCORE von PacketVideo, unterstützt MPEG4, H.264, MP3, AAC, AMR, JPEG und PNG.
- **OpenGL|ES**: 3D-Graphiken für Embedded Systeme.
- **SGL**: 2D Graphik-Engine.
- **SQLite**: Schlankes, relationales Datenbanksystem.
- **SSL**: Für SSL-basierte Sicherheit für Netzwerkkommunikation.
- **Surface Manager**: Bibliothek zur Verwaltung der Zugriffe auf das Display-Subsystem.

# Android-Architektur

---

## Android-Runtime

- Kernbibliotheken für die Laufzeitumgebung - Eine Teilmenge von Apache Harmony Java 5 Implementierung. 
- **Dalvik Virtuelle Maschine** (DVM), ab Lollipop **Android-Runtime** (ART): registerbasierte Maschine. DVM interpretiert Dateien in Dalvik Executable (**dex**)-Format. ART wandelt den Bytecode während der Installation einer Anwendung in den nativen Binärcode um (odex) => schnelle Ausführung.
- Jede Applikation läuft im separaten Prozess und **hat eine eigene** DVM/ART.
- Thread-Modell und Low-Memory-Verwaltung von Linux.


## Linux-Kernel

- Abstraktionsschicht zwischen Android-Welt und Hardware.
- Linux bietet das Sicherheitsmodell, Prozess- und Speicherverwaltung, Netzwerkstack und Treiber.

# Android-Architektur

---



## App-Architektur

- Android-App-Architektur basiert auf Komponenten, die untereinander mit Hilfe von **Intents** kommunizieren.
- Eine App ist eine Sammlungen von Komponenten (**Activities**, **Services**, **Content Providers** und **Broadcast Receivers**). Nicht jeder Typ muss in einer App vorhanden sein.
- Jede App läuft in ihrem eigenen Linux-Prozess. Android startet einen  Prozess, wenn eine Komponente der App benötigt wird. Es gibt kein **main** wie in Java oder C/C++.
- Komponenten einer App teilen gemeinsam eine Menge von Ressourcen (Datenbanken, Präferenzen, Dateisystem).

# Android-Architektur

---


## Wichtige Bestandteile

- **Activity:**
  - Sichtbarer Teil einer Anwendung zur Interaktion mit dem Benutzer
  - hat einen eigenen Lebenszyklus.
  - Eine App hat in der Regel mehrere Activities.
- **Service:** Läuft im Hintergrund (wie Windows-Service bzw. Linux-Dämon).
  - **local service:** Läuft in demselben Prozess wie der Rest der App. 
  - **remote service:** Läuft in einem separaten Prozess, wird über Interprozesskommunikation erreicht. 
- **Content Providers:** Bereitstellung von Daten für andere Apps.
- **Broadcast Receivers:** Können Systemereignisse empfangen und darauf reagieren.

# Entwicklungsumgebung

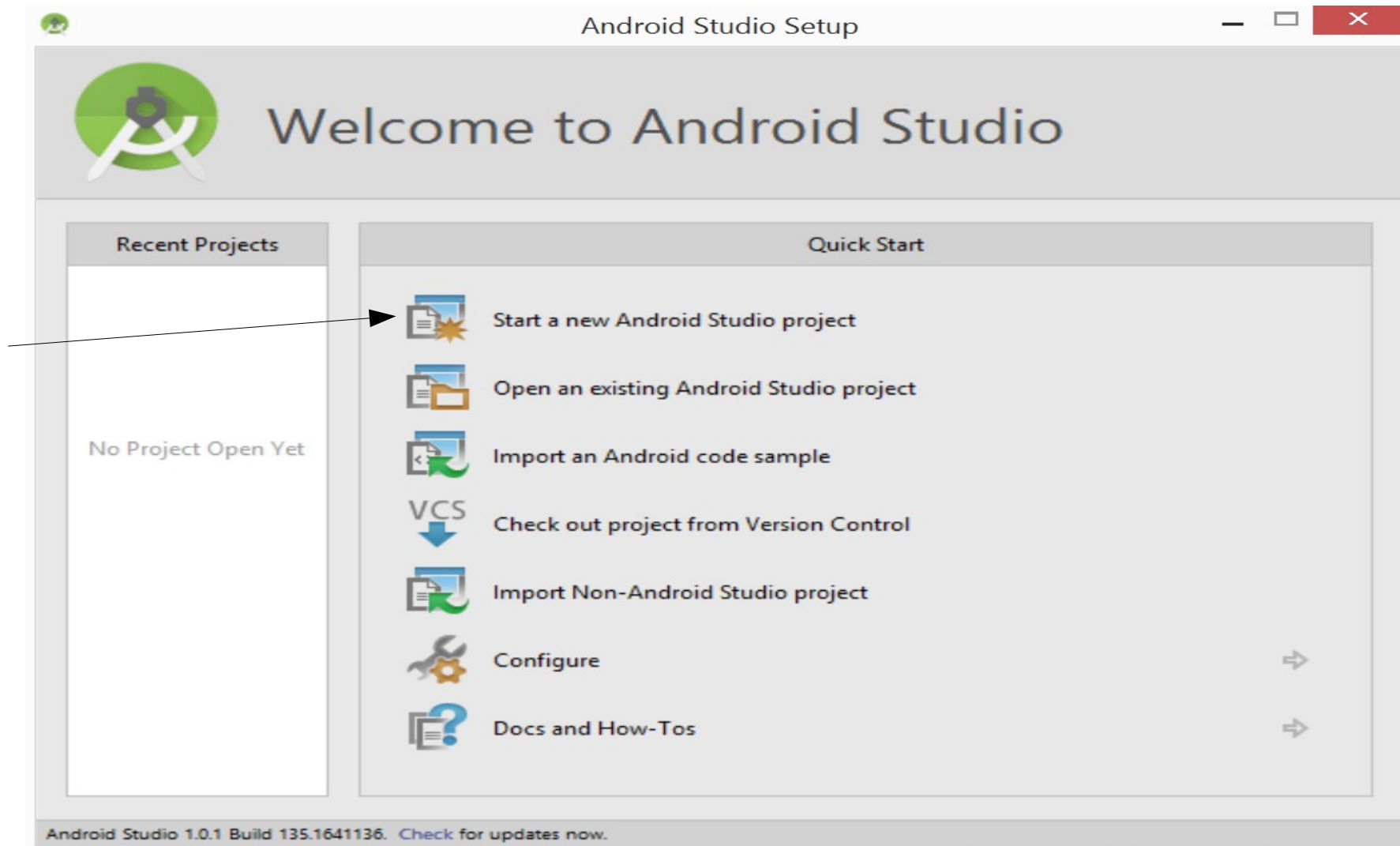
---

Im Rahmen der Vorlesung wird Android-Studio verwendet. Benötigt werden

- Windows, Mac OS X oder Linux
- Mindestens JDK 5 (JDK 8 nicht unterstützt, JDK 7 nur teilweise)
- Android-Studio (Download von <http://developer.android.com/sdk/index.html>)
- Unter Windows müssen ggf. USB-Treiber für die Geräte vom Hersteller installiert werden. 

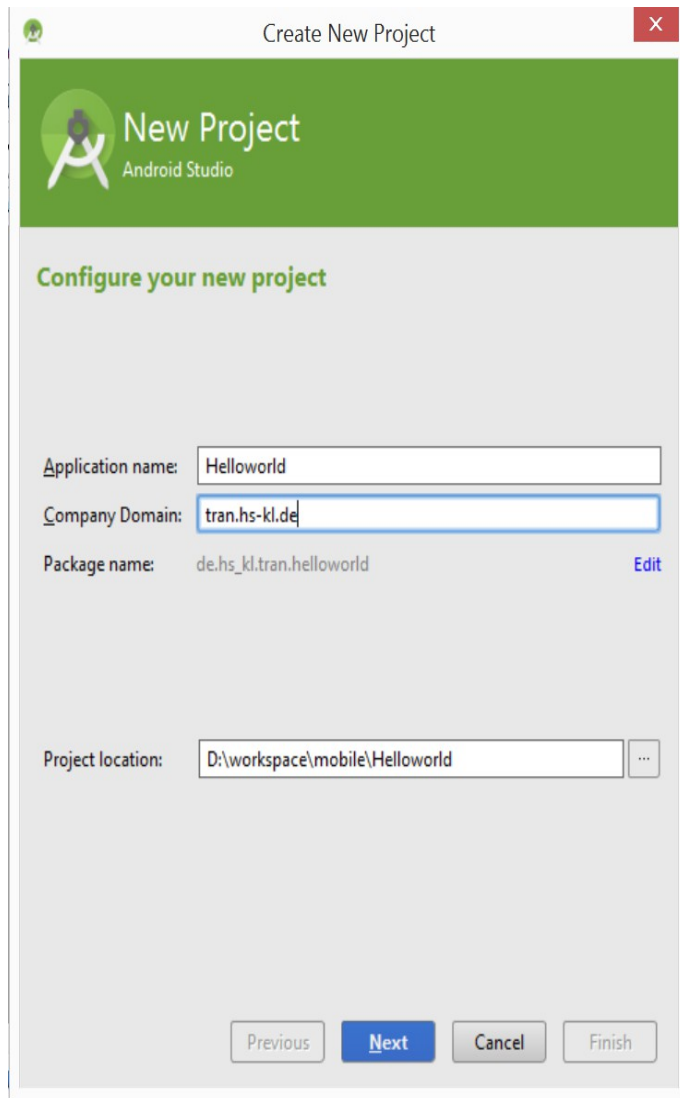
Alternativ: Eclipse mit ADT-Plugin (nicht mehr von Google unterstützt)

# Erstes Beispiel





# Erstes Beispiel



**Create New Project**

**New Project**  
Android Studio

**Configure your new project**

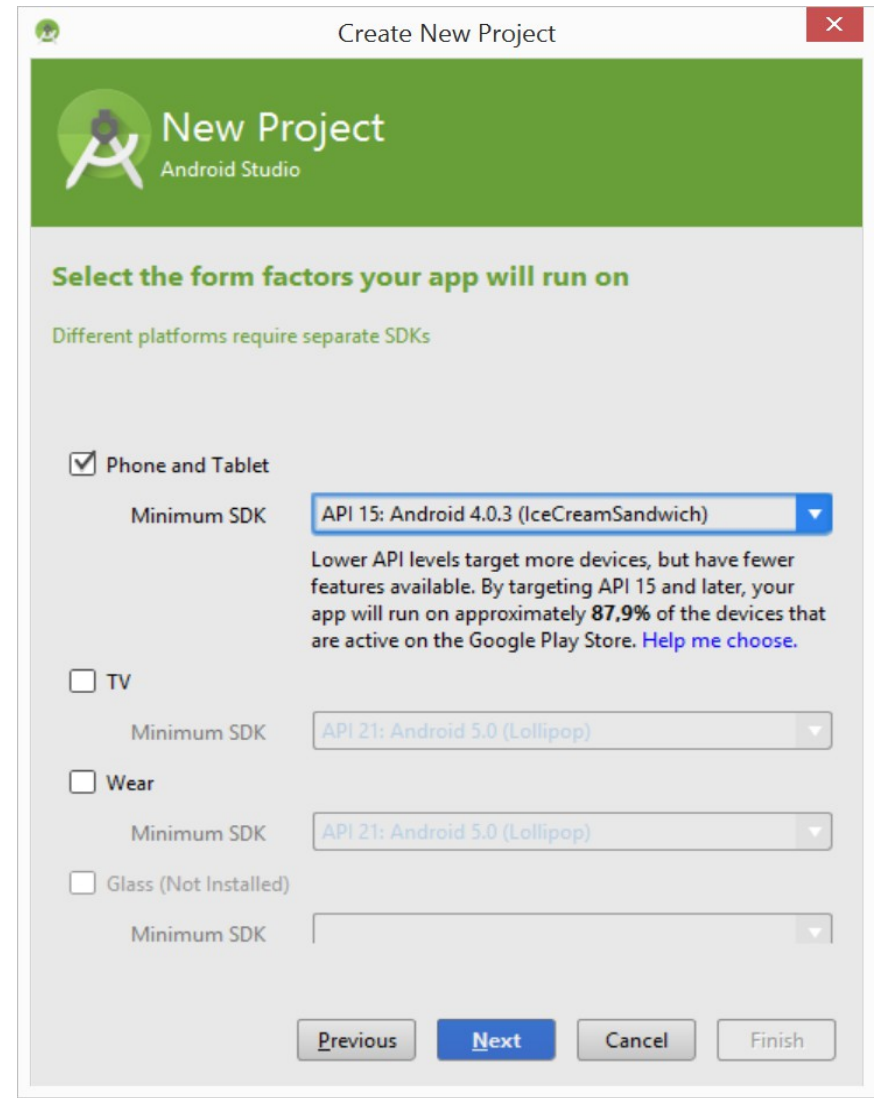
Application name:

Company Domain:

Package name:  [Edit](#)

Project location:  [...](#)

[Previous](#) [Next](#) [Cancel](#) [Finish](#)



**Create New Project**

**New Project**  
Android Studio

**Select the form factors your app will run on**

Different platforms require separate SDKs

☒ **Phone and Tablet**

Minimum SDK:  [v](#)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately **87,9%** of the devices that are active on the Google Play Store. [Help me choose.](#)

☐ **TV**

Minimum SDK:  [v](#)

☐ **Wear**

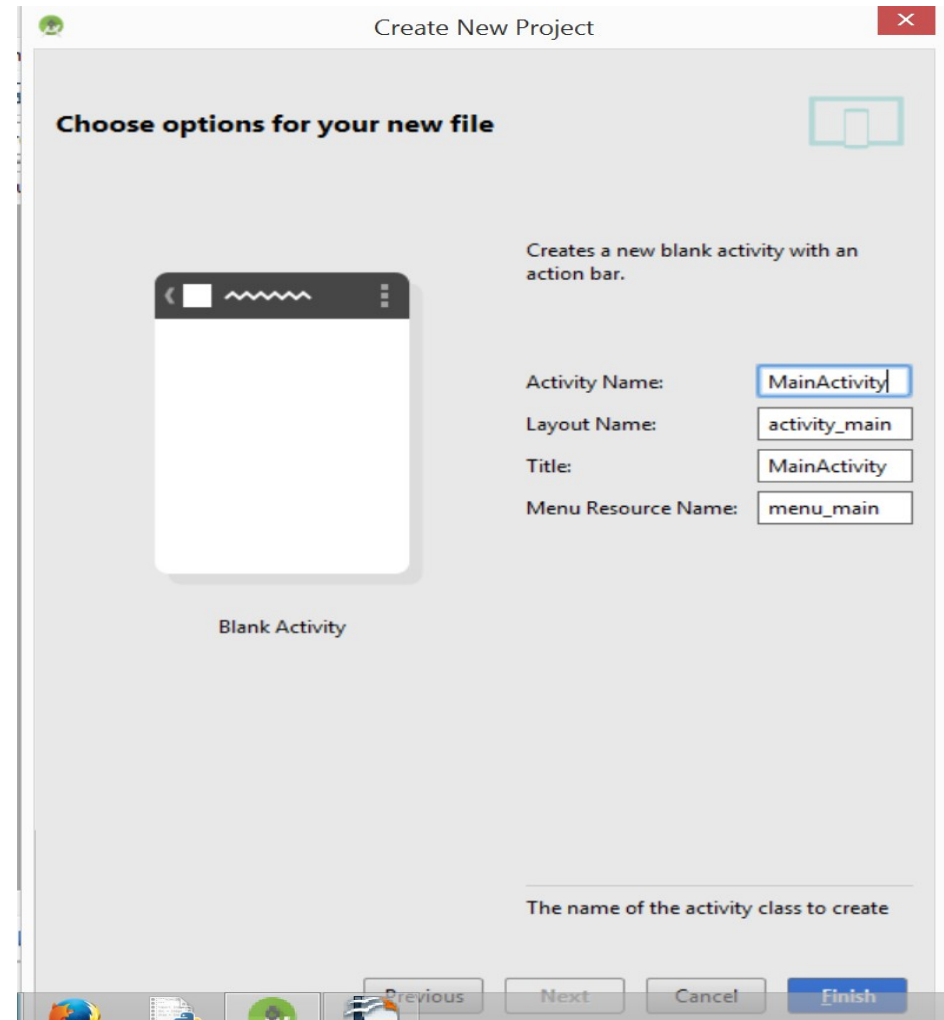
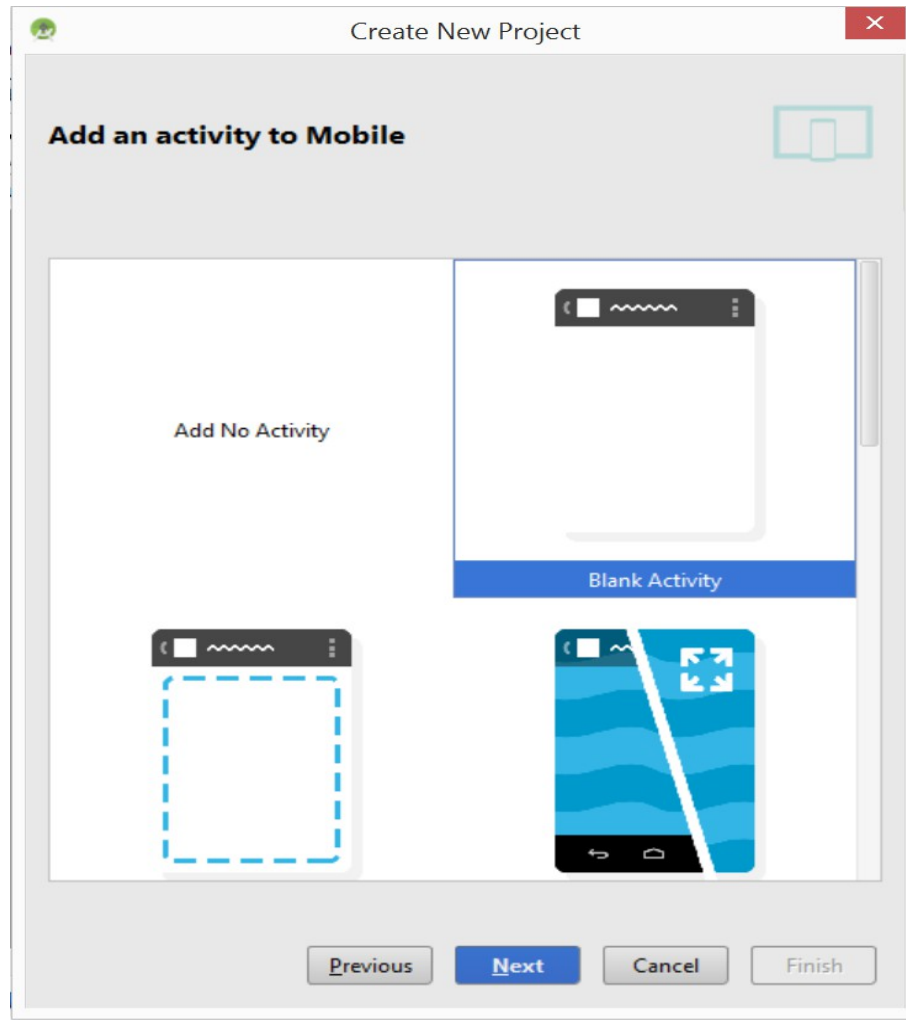
Minimum SDK:  [v](#)

☐ **Glass (Not Installed)**

Minimum SDK:  [v](#)

[Previous](#) [Next](#) [Cancel](#) [Finish](#)


# Erstes Beispiel



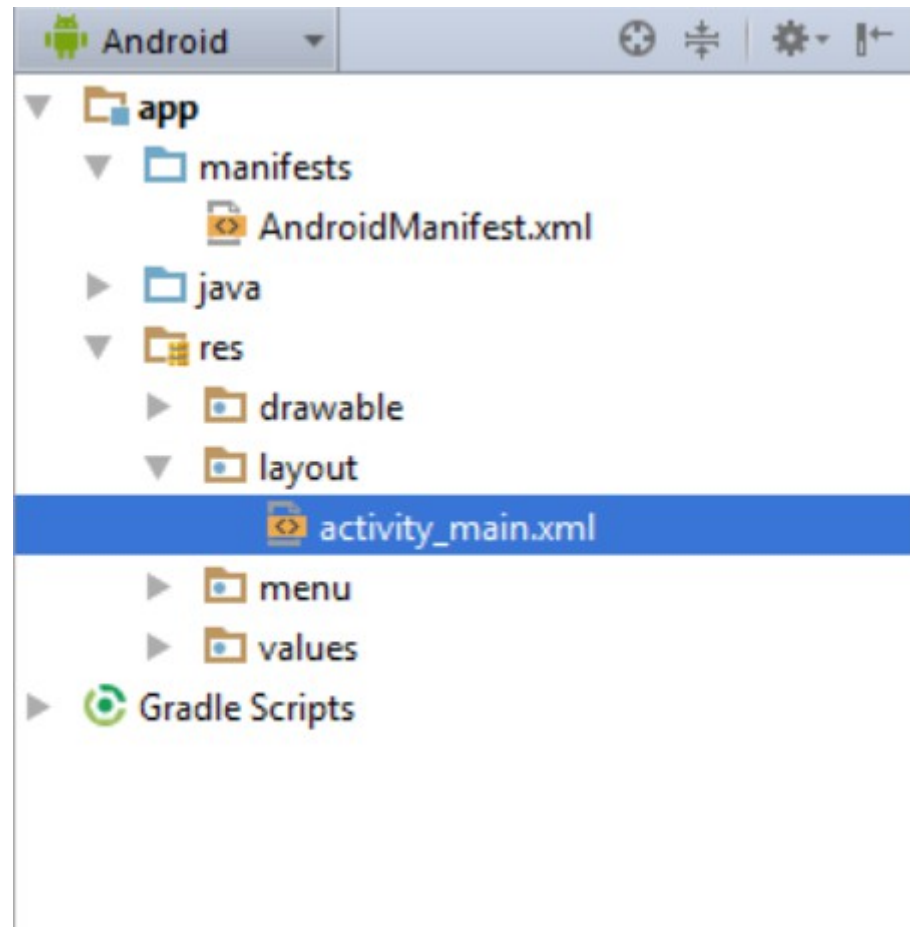
# Erstes Beispiel

## Projektstruktur (unter der Ansicht Android)

### AndroidManifest.xml:

XML-Datei zur  Beschreibung der Anwendung (Komponenten, Erlaubnis, Filter usw.).

**Gradle Scripts:** werden automatisch erzeugt, für die verschiedenen Plattformen, Abhängigkeiten.



## Erstes Beispiel

---

- Ordner **java**: Für unsere erstellten Klassen.
- Ordner **res**: Ressourcen, wie Strings, Graphiken, Layout-Definitionen usw. werden in die App gebunden (nicht in Roh-Form – Der Compiler wandelt sie um)

Ressourcen (Non-Code-Daten) werden indiziert. Jeder dieser Ressource hat einen Index in der Datei `R.java` und kann über den Index zur Laufzeit zugegriffen werden (siehe später).

Für Details über die Projektstruktur siehe

<https://developer.android.com/tools/studio/index.html#project-structure>

# Erstes Beispiel

## Erzeugtes AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="de.hs_kl.tran.helloworld" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Beschreibung  
der Activity

Verweis auf Ressourcen

# Erstes Beispiel

---

## Erzeugte Activity

```
package de.hs_kl.tran.helloworld;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

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

Die View ist in `res/layout/activity_main.xml` definiert und wird hier benutzt.

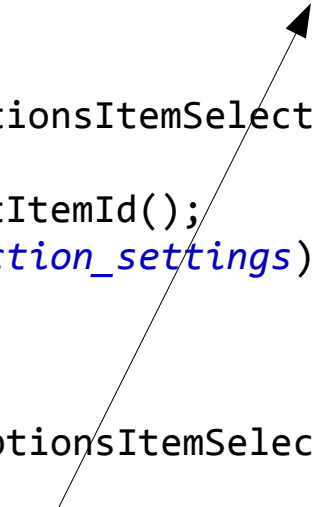
# Erstes Beispiel

---

## Fortsetzung

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    if (id == R.id.action_settings)
    {
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```



Das Menü ist in `res/menu/main.xml` definiert und wird hier benutzt.

# Erstes Beispiel

---

## Generierte R.java (Auszug – Ansicht Project – app\build\generated\...)

```
public final class R {  
    ...  
    public static final class drawable {  
        ...  
        public static final int ic_launcher=0x7f020000;  
    }  
    public static final class id { ...  
        public static final int action_settings=0x7f080000;  
    }  
    public static final class layout { ...  
        public static final int activity_main=0x7f030000;  
    }  
    public static final class menu { ...  
        public static final int main=0x7f070000;  
    }  
    public static final class string {  
        ...  
        public static final int action_settings=0x7f050001;  
        public static final int app_name=0x7f050000;  
        public static final int hello_world=0x7f050002;  
    }  
    ...  
}
```



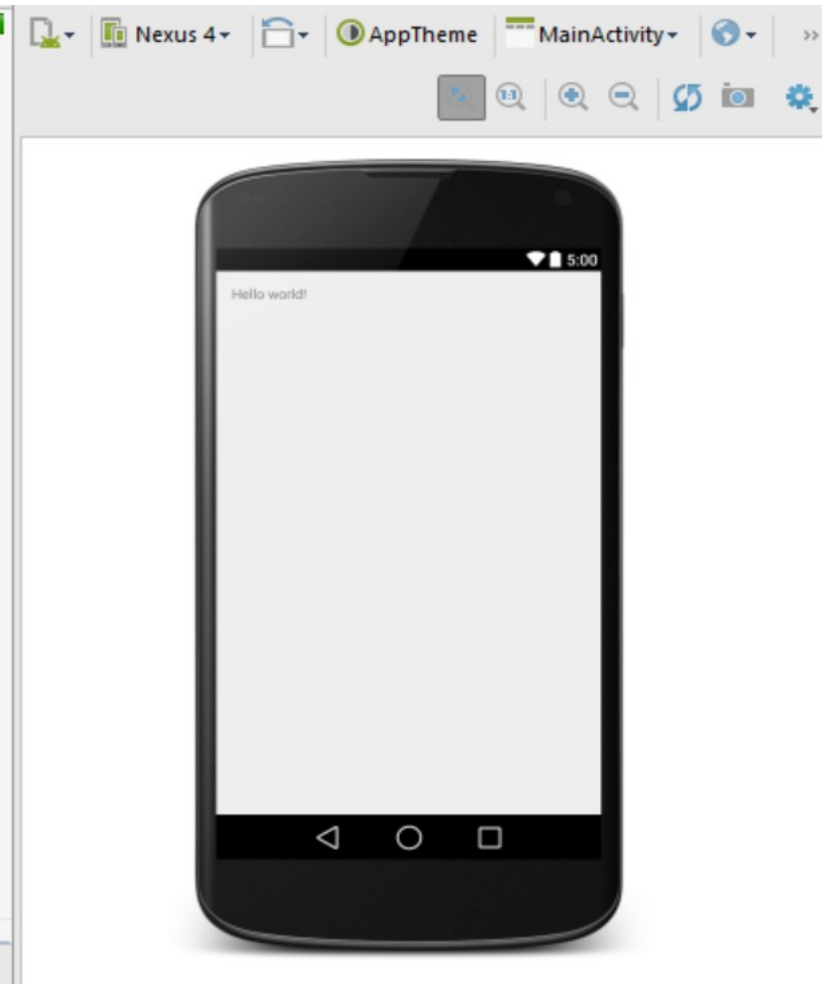
# Erstes Beispiel

## Erzeugtes activity\_main.xml

```
<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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView android:text="Hello world!" android:layout_width="wrap"
        android:layout_height="wrap_content" />

</RelativeLayout>
```



# Erstes Beispiel

---

## Erzeugtes strings.xml

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

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

</resources>
```

Und weiter Dateien wie styles.xml, dims.xml, menu/main.xml usw.

## Erstes Beispiel

---

**Android Virtual Device (AVD):** Ein AVD stellt eine Gerätekonfiguration dar (einen geeigneten Emulator).

Tools -> Android-> AVD Manager. dann New... wählen und entsprechendes AVD erzeugen.

Besser: Testen mit einem realen Gerät. Dazu wird unter Windows der passende USB-Treiber benötigt.

# Erstes Beispiel

---

## **Helloworld-App in Emulator**

Alternativ: Genymotion



## Zweites Beispiel

---

Project name: BrowserIntent

Danach sollen folgende Änderungen durchgeführt werden:

### **res/layout/strings.xml**

Eine neue Zeichenkette wird definiert (bequemer Weg ist die Benutzung des Ressourcen-Editors).

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

    <string name="app_name">BrowserIntent</string>
    <string name="action_settings">Settings</string>
    <string name="EDIT_TEXT_HINT">enter URL here</string>
    <string name="OK_TEXT">OK</string>

</resources>
```

## Zweites Beispiel

### res/layout/activity\_browser\_intent.xml (mit dem Layout-Editor üben!)

```
<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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".BrowserIntentActivity">
```

```
    <LinearLayout
```

```
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
```



```
        <EditText
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/EDITTEXT"
            android:layout_weight="1"
            android:hint="@string/EDIT_TEXT_HINT" />
```



```
        <Button
```

```
            android:layout_width="97dp"
            android:layout_height="wrap_content"
            android:text="@string/OK_TEXT"
            android:id="@+id/OK_BUTTON" />
```

```
    </LinearLayout>
```


```
</RelativeLayout>
```




## Zweites Beispiel

---

### BrowserIntentActivity.java

```
public class BrowserIntentActivity extends ActionBarActivity implements
OnKeyListener, OnClickListener 
{
    private EditText mEditText;
    private Button mButton;

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

        // get handle to the GUI-elements
        mEditText = (EditText) findViewById(R.id.EDITTEXT); 
        mButton = (Button) findViewById(R.id.OK_BUTTON);
        // setup event handler
        mEditText.setOnKeyListener(this);
        mButton.setOnClickListener(this);
    }
}
```

## Zweites Beispiel

---

### Callbacks

```
// implementation for onKeyListener
public boolean onKey(View view, int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_ENTER) {
        openBrowser();
        return true;
    }
    return false;
}

// implementation for OnClickListener
public void onClick(View v) {
    openBrowser();
}
```



## Zweites Beispiel

---

```
/**
 * open a browser on the URL specified in the text box
 */
private void openBrowser() {
    String txt = mEditText.getText().toString();
    if (!(txt.startsWith("http://") || txt.startsWith("https://"))) {
        txt = "http://" + txt;
    }
    Uri uri = Uri.parse(txt);
    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    startActivity(intent);
}
```

# Debugging-Ausgabe

---


Debugging mit `android.util.Log`-Klassenmethoden:

- `Log.e()`: Fehler – schwer wiegende Fehler.
  - `Log.w()`: Warnungen – Zum Protokollieren unerwarteter Verhalten.
  - `Log.i()`: Informationen – Meist Informationen über geladene Modulen.
  - `Log.d()`: Debugging - Debug-Information
  - `Log.v()`: Verbose (gesprächig)
- 
- Parameter 1: Quelle (z.B. – Klassenname),
  - Parameter 2: Die Meldung.

# Debugging-Ausgabe

---

## Beispiel

```
private static final String TAG = 
MyActivity.class.getSimpleName();
```

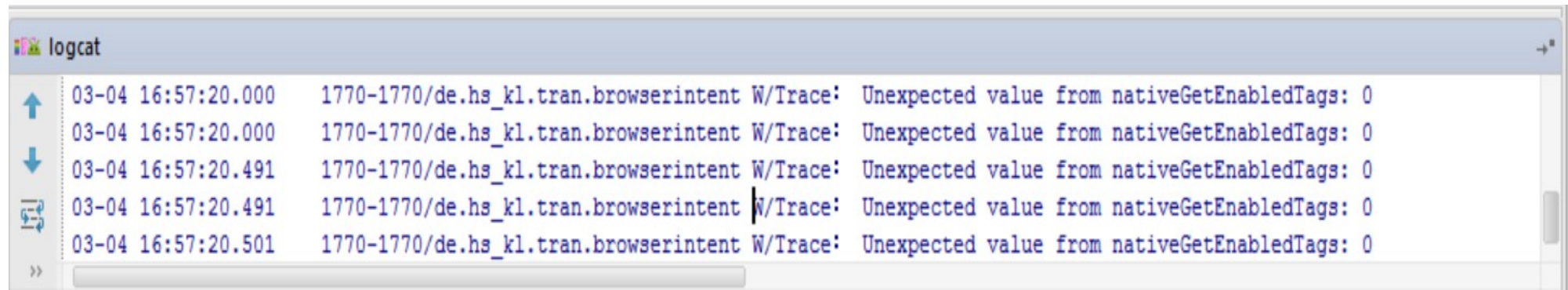
```
..
```

```
Log.v(TAG, "index=" + i);
```

- Verbose-Ausgaben werden nur während der Entwicklung in die Applikation eingebunden (=> Codeersparnis).
- Debug-Ausgaben werden kompiliert aber zur Laufzeit unterdrückt (nur Ausgabe, wenn USB-Debugging aktiviert wird => sparsame Anwendung!).
- Logging-Ausgaben können unter `LogCat` angesehen werden (USB-Debugging-Option auf dem Geräte muss über Einstellung **enabled** werden – Im Emulator nicht notwendig).
- Totale Eliminierung mit ProGuard möglich => siehe Literatur

# Debugging-Ausgabe

---



logcat

```
03-04 16:57:20.000 1770-1770/de.hs_kl.tran.browserintent W/Trace: Unexpected value from nativeGetEnabledTags: 0
03-04 16:57:20.000 1770-1770/de.hs_kl.tran.browserintent W/Trace: Unexpected value from nativeGetEnabledTags: 0
03-04 16:57:20.491 1770-1770/de.hs_kl.tran.browserintent W/Trace: Unexpected value from nativeGetEnabledTags: 0
03-04 16:57:20.491 1770-1770/de.hs_kl.tran.browserintent W/Trace: Unexpected value from nativeGetEnabledTags: 0
03-04 16:57:20.501 1770-1770/de.hs_kl.tran.browserintent W/Trace: Unexpected value from nativeGetEnabledTags: 0
```

Log level: Verbose ▼ Q Main ✕



```
V/MainActivity: index=http://www.orf.at
V/MainActivity: index=http://www.orf.at
```

---

# **Android-Manifest**

# Android Manifest

---

- Jede Android-Anwendung muss eine `AndroidManifest.xml` für wichtige Information über Anwendungsidentität, -namen, Versionsnummer, Beschreibung der Komponenten, Erlaubnisse usw. haben.
- Das Laufzeitsystem verwendet sie, um (unter anderen)
  - Anwendung zu installieren und zu updaten,
  - Anwendungsdetails (Name, Icon, Beschreibung) anzuzeigen,
  - Erlaubnisse zu überwachen/verwalten,
  - Anwendungsactivities zu starten,
  - Entwickler zu erlauben, Anwendung zu debuggen.

# Android Manifest

---

Einige wichtige Bestandteile sind (für weitere siehe [5] – teilweise unter Android Studio ausgelagert in build.gradle):

## Applikationsidentität

- Paketname, Versionscode, Versionsname.
  - Paketname muss sorgfältig gewählt werden. Er ist die Identifikation Ihrer Anwendung, kann nach der Veröffentlichung nicht mehr geändert werden.
  - Versionscode wird verwendet für Updates.
  - Versionsname wird den Benutzern angezeigt.

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

# Android Manifest

---

## System Requirements

- SDK-Version:
  - **minSdkVersion**: Der kleinste unterstützte API-Level.
  - **targetSdkVersion** (optional): Der optimal unterstützte API-Level
  - **maxSdkVersion** (optional): Der größte unterstützte API-Level.

```
<uses-sdk android:minSdkVersion="14" />
```



- Anwendungsname und -icon (zur Anzeige). Optional kann eine Beschreibung und das Einschalten des Debugging stehen:

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:description="@string/app_desc"
    android:debuggable="true"
>
```





# Android Manifest

---

## Plattform Requirements

- Externe Bibliothek (z.B. um Google-API zu verwenden). Standard sind die Pakete `android.app`, `android.content`, `android.view` und `android.widget` in der Standardbibliothek. Benötigt die Anwendung spezielle Pakete (z.B. `maps`), dann muss die passende Bibliothek eingebunden und angegeben werden:

```
<application ...>
    ...
    <uses-library android:name="com.google.android.maps">
    </uses-library>
    ...
</application>
```

# Android Manifest

---

## Activities und andere Komponenten registrieren

- Eine Liste von Activities, Services, Content Providers und Broadcast Receivers.
- Alle Komponenten müssen registriert werden. Sonst werden sie nicht gestartet.
- Activities , die nur in der Applikation verwendet werden, haben eine einfache Beschreibung:

```
<activity android:name=".weitereActivity" > </activity>  
<activity android:name=".undNochEineActivity" > </activity>
```

- Der Punkt vor dem Namen kann entfallen. Er verleiht nur den Nachdruck, dass die Activity zu dem Paket gehört.
- Activities, Services und Broadcast Receivers, die von anderen Anwendungen bzw. vom System gestartet werden sollen, haben weitere Beschreibungen im Block `<intent-filter> </intent-filter>` (siehe später).

# Android Manifest

---

- Eine Activity muss als Eintrittspunkt der Anwendung definiert werden.

```
<activity
    android:name=".ManifestDemoActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

## Erlaubnisse (Permissions)

- Android-Anwendung hat in der Regel kein Erlaubnis, auf andere Ressourcen zuzugreifen. Benötigt die Anwendung dennoch Zugriffe darauf, muss der Benutzer bei der Installation zustimmen. Erbetene Erlaubnisse müssen angegeben werden (für die Konstanten siehe auch [android.Manifest.permission](#))

```
<permission android:name="android.permission.CAMERA" />
<permission android:name="android.permission.READ_CONTACTS" />
<permission android:name="android.permission.WRITE_CONTACTS" />
```

---

# Ressourcen

# Ressourcen

---

- Zum Programm gehören auch Graphiken, Icons, Klangdateien, Videos, Texte in verschiedenen Sprachen.
- Farbdefinitionen, Menüeinträge, Listen, Layouts usw. werden in Android häufig nicht in Code definiert. Sie werden in XML als Ressourcen hinterlegt.
- Ordnerstruktur für Ressourcen ist streng definiert:
  - Alle Ressourcen liegen in einem Unterordner von `/res`.
  - Für Ordner- und Dateinamen darf **nur Kleinschreibung oder Unterstrich** verwendet werden (Ausnahme: Länderkürzel wie z.B. DE; US).
  - Verschiedene Ressourcen müssen in verschiedenen Unterordnern liegen.
  - Mit Ausnahme vom Ordner `assets` dürfen andere Unterordner keinen Unterordner beinhalten.
- Ressourcen (Ausnahme: Daten in `assets`) werden kompiliert und können über ID angesprochen werden.

# Ressourcen

Ressourcentyp	Unterordner	Dateiname	XML Tag
String	res/values	strings.xml (empfohlen)	<string>
String-Mehrzahl	res/values	strings.xml (empfohlen)	<plurals>, <item>
String-Array	res/values	strings.xml (empfohlen)	<string-array>, <item>
Wahrheitswerte	res/values	bools.xml (empfohlen)	<bool>
Farben	res/color	colors.xml (empfohlen)	<color>
Farbenliste (für Zustände)	res/color	Bsp. buttonstates.xml indicators.xml	<selector>, <item>
Größen	res/values	dimens.xml	<dimen>
Integers	res/values	integers.xml (empfohlen)	<integer>
Integer-Arrays	res/values	integers.xml (empfohlen)	<integer-array>, <item>
Mixed-Type Arrays	res/values	arrays.xml	<array>, <item>
Einfache Drawables	res/values	drawables.xml (empfohlen)	<drawable>

# Ressourcen

Ressourcentyp	Unterordner	Dateiname	XML Tag
Bilder	res/drawable	Bsp. sun.jpg, logo.png	Graphikdateien oder XML-Drawable- Definition
Tweening	res/anim	Bsp. animsequence.xml	<set>, <alpha>, <scale>, <rotate>, <translate>
Frame-by-Frame Animation	res/drawable	Bsp. sequence1.xml	<animation-list>, <item>
Menü	res/menu	Bsp. meinmenu.xml, abc.xml	<menu>, <item>
XML Dateien	res/xml	Bsp data1.xml, data2.xml	benutzerdefiniert
Layouts	res/layout	Bsp. main.xml, demoscreen.xml	Siehe Kapitel über Views
Raw	res/raw	Bsp. jingle.mp3, hello.mp4, help.txt	benutzerdefiniert
Styles und Themes	res/values	styles.xml, themes.xml (empfohlen)	<style>

# Zugriff auf Ressourcen

---

Es gibt 3 Möglichkeiten, auf Ressourcen zuzugreifen

- per Referenz innerhalb der Ressourcendefinition,
- per Referenz im Code,
- per Direktzugriff im Code.

## Referenz innerhalb der Ressourcendefinition

Zugriff (mit dem @-Zeichen) über einen Ressourcenschlüssel der Form

**@ [package: ] ressourcenart/ressourcenname**

```
<application
    android:debuggable="true"
    android:description="@string/app_desc"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Light" >
```



# Zugriff auf Ressourcen

---

## Referenz innerhalb des Java-Code

Zugriff über Konstanten von `R.java`. Zum Beispiel

```
public boolean onOptionsItemSelected(MenuItem item) {  
    int id = -1;  
    int titleId = -1;  
    switch (item.getItemId())  
    {  
        case R.id.absolute_menu_item:  
            id = R.layout.absolute_layout;  
            titleId = R.string.absolutLayout;  
            ....  
            // weiterer Code  
  
            if (id != -1) [  
                setContentView(id);  
                setTitle(titleId);  
                return true;  
            ]  
    }
```

**Vorsicht:** Konsistenz (richtige ID für das richtige Objekt) kann erst zur Laufzeit geprüft werden (Absturz möglich!).

# Zugriff auf Ressourcen

---

## Datenzugriff innerhalb des Java-Codes

Zugriff über `getResources` und über `R.java`. Zum Beispiel

```
String title = getResources().getString(R.layout.absolute_layout);  
Drawable logo = getResources().getDrawable(R.drawable.robot);
```

## Alternative Ressourcen

Android erlaubt alternative Ressourcenordner für verschiedene Konfigurationen. Zum Beispiel

```
\res\layout\main_layout.xml           // Default  
  
\res\layout-land\main_layout.xml      // für Landscape-Mode  
  
\res\layout-car\main_layout.xml       // Car-Modus  
  
\res\values\strings.xml               // Default  
  
\res\values-en\strings.xml           // Englisch  
  
\res\values-en-rUS\strings.xml        // USA  
  
\res\drawable (320x480 dpi), \res\drawable-lpi (240x320 dpi),  
\res\drawble-hpdi (480x800 dpi)
```

# Ressourcen

---

## String

Syntax `<string name="ressourcenname">wert</string>`

Im Code `getResources().getString(R.string.ressourcenname);`

Verwendung von `<b></b>` (bold), `<i></i>` (italic) `<u></u>` (underline) möglich.  
Vor einem doppelten Anführungszeichen wird das Escape-Zeichen `\` benötigt.  
Bei einfachen Anführungszeichen ist es nicht notwendig. Für Behandlung HTML-Tags im Code siehe Literatur.

## Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello"><b>Hello World!</b></string>
    <string name="app_name">"ManifestDemo"</string>
    <string name="app_desc">Das ist ein \" wichtiger \" Test</string>
    <string name="einfach">ein einfacher Test</string>
    <string name="quote_ok1">ein \'einfacher\' Test</string>
    <string name="quote_ok2">"ein 'einfacher' Test"</string>
    <string name="formatString" formatted = "false">Score: %1$d of %2$d</string>
</resources>
```

# Ressourcen

---

## String-Array

Syntax    `<string-array name="ressourcenname">`  
              `{ <item>text</item> }+`  
              `</string-array>`

Im Code    `getResources().getStringArray(R.array.ressourcenname)`

## Beispiel

```
<string-array name = "vorspeise">
    <item> Suppe </item>
    <item> Frühlingsrolle </item>
    <item> Salat </item>
</string-array>
```

```
String[] vorspeise =
    getResources().getStringArray(R.array.vorspeise);
```

# Ressourcen

---

## String-Mehrzahl

**Syntax** `<plurals name="plurals_name">`  
    `<item quantity = "one" >text</item>`  
    `<item quantity = "other" >text (mit %d)</item>`  
`</plurals>`

Erlaubt sind z.B. zero, one, two, other

**Im Code** `getResources()` .

`getQuantityString(R.plurals.plurals_name,int id, int subst.)`

## Beispiel

```
<plurals name = "plurals_test">
    <item quantity = "one"> 1 Ei</item>
    <item quantity = "other">%d Eier</item>
</plurals>
```

```
String s = getResources().getQuantityString(R.plurals.plurals_test, 5, 5);
```

# Ressourcen

---

## Boolean

Syntax `<bool name="bool_name"> true|false </bool>`

Im Code `getResources().getBoolean(R.bool.bool_name)`

## Integer

Syntax `<integer name="int_name">wert</integer>`

Im Code `getResources().getInteger(R.integer.int_name)`

## Integer-Array

Syntax `<integer-array name="intarray_name">`

`{<item> wert <item>}+`

`</integer-array>`

Im Code `getResources().getIntArray(R.array.intarray_name)`

# Ressourcen

---

## Farben

Folgende Formate sind erlaubt

- #RGB (Bsp. #F00 12-bit Farbe, rot)
- #ARGB (Bsp. #80F0 12-bit Farbe, grün mit 50% Alpha)
- #RRGGBB (Bsp. #FF00FF 24-bit Farbe, violett)
- #AARRGGBB (Bsp: #80FF00FF)

Syntax    <color name="color\_name"> Wert </color>

Im Code    getResources().getColor(R.color.color\_name)

### Beispiel

```
<color name="startColor">#00FF00</color>
<color name="endColor">#001100</color>
<color name="yellow">#aaaa00</color>
```

# Ressourcen

---

## Größen

Android unterstützt folgende Einheiten:

- **px** (Pixel): Punkt (dot) auf dem Bildschirm,
- **in** (Inch): (2,54 cm),
- **mm** (Millimeter),
- **pt** (Point): 1 Inch = 72 Points,
- **dp** bzw. **dip** (density-independent pixel): Eine abstrakte Einheit basiert auf die Dichte des Bildschirms. Auf einem Display mit 160 Punkten pro Inch gilt  $1_{dp} = 1_{px}$ ,
- **sp** (scale-independent pixel): Ähnlich wie **dp**, wird für Font-Größe verwendet.

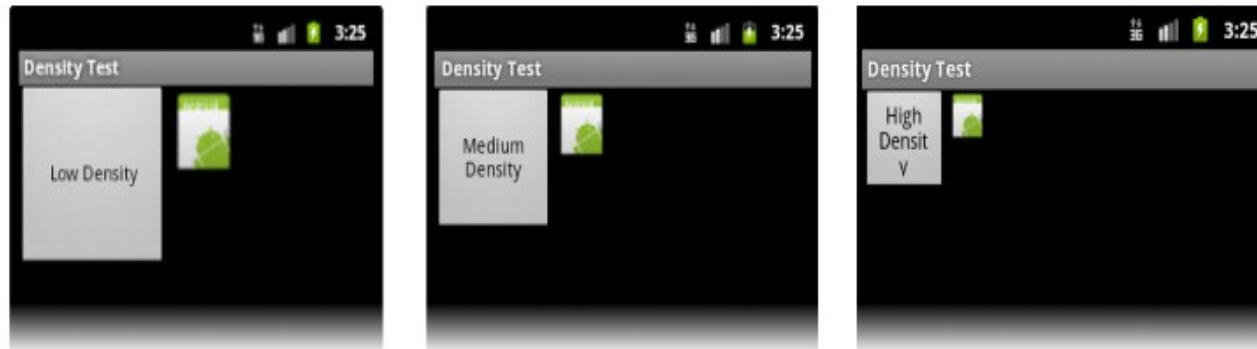
Um Anwendungen für verschiedene Displays zu realisieren, sollten nur **sp** und **dp** (**dip**) verwendet werden.



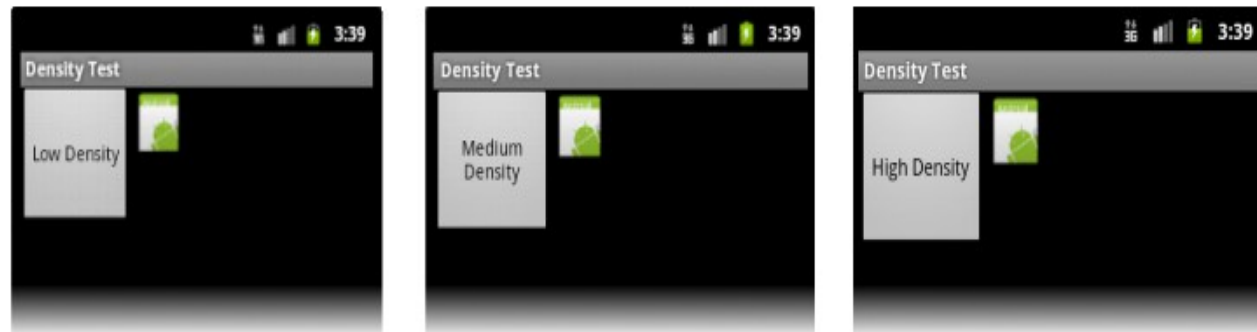
# Ressourcen

---

Angaben in Pixels bei verschiedenen Auflösungen können dazu führen:



Unter der Verwendung von Einheit  $dp$ :



# Ressourcen

---

## Größen

Syntax `<dimen name="groesse_name"> Wert [Einheit] </dimen>`

Im Code `getResources().getDimension(R.dimen.groesse_name)`

## Einfache Drawables

Syntax `<drawable name="rect_name"> Color-Wert </drawable>`

Im Code `getResources().getDrawable(R.drawable.rect_name)`

Einfache graphische Objekte wie Farbrechtecke werden unter dem Ordner `/res/values` definiert.

## Beispiel

```
<resources>
```

```
    <drawable name="red_rect">#F00</drawable>
```

```
    <drawable name="yellow_rect">#0FF</drawable>
```

```
</resources>
```

# Ressourcen

---

Komplexere Formen werden in `/res/drawable` zusammen mit anderen Bildern abgelegt.

## Beispiel

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval" >

    <gradient

        android:startColor="@color/startColor"

        android:endColor="@color/endColor"

        android:type="linear"

        android:angle="270"/>

</shape>
```

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

# Ressourcen

---

## Bilder

Android unterstützt die Bildformate `png`, `jpeg`, `gif` und Nine-Patch (`.9.png`).

Nine-Patch ist ein spezielles Format, bei dem man skalierbare Bereiche angeben kann. Mit `draw9patch` unter `/tools` vom Android SDK kann man Nine-Patch-Bilder erzeugen.

Im Code `getResources().getDrawable(R.drawable.file_name)`

## Audiodateien

können unter `/res/raw` abgespeichert. Man kann die ID der Datei an den MediaPlayer zum Abspielen weitergeben

```
MediaPlayer p = MediaPlayer.create(this, R.raw.sleepaway);  
p.start();
```

# Ressourcen

---

## Videodateien

können unter `/res/raw` abgespeichert. Man kann die Movie über

```
Movie v = getResources().getMovie(R.raw.myfilm);
```

Einlesen.

## XML-Dateien

Sollen XML-Dateien nicht kompiliert werden, legt man sie im `/res/raw`. Mit `getXml` kann man ein `XmlResourceParser`-Objekt erhalten und kann sie selbst behandeln.

**Animation** (siehe Literatur)

**Menü, Button-Zustände, Layout, Stile und Themen** siehe später

# Ressourcen

---

## Der assets-Ordner

Der `assets`-Ordner wird selten verwendet. Hier können Ressourcen aller Art abgelegt werden. Das Laden der Ressourcen geht über den **AssetManager**. Für diesen Ordner gilt:

- variable Ordnerstruktur,
- keine Indizierung der Ressourcen,
- Zugriff nur über den Dateipfad (man öffnet dafür einen `InputStream`),

```
InputStream f = null;
try
{
    f = getAssets().open("/data/text.txt");
}
catch (IOException ex) {}
```

- Ressourcen werden nicht vor-kompiliert.