

# Android – Eine Einführung

## Layouts, Views & Adapter

Andreas Wilhelm

CSC Computer-Schulung & Consulting GmbH

# Contents

1. Layouts
2. View-Klassen
3. ViewGroup-Klassen
4. AdapterView-Klassen
5. Adapter

# Layouts

# Contents

- 1 Überblick
- 2 ViewGroups
- 3 Views
- 4 RequestFocus
- 5 Include
- 6 Merge

## Allgemeines

- ▶ Oberflächen für Aktivitäten, Menüs, Dialoge und Widgets
- ▶ Grundelemente Views und ViewGroups (Layouts & AdapterViews)
- ▶ Deklaration im XML-Format
- ▶ Formatierung eines Elements über Attribute
- ▶ Alternativ View-Objekte während der Laufzeit im Quellcode erzeugen und ändern
- ▶ Weitestgehend einheitliche Namensgebung für XML-Attribute und Objekt-Methoden

### XML-Layouts

Deklaration von Layouts in XML verschlankt den Code, ermöglicht Unterstützung verschiedener Oberflächen, Sprachen und Auflösungen und erleichtert den Debugging-Prozess.

## Deklaration

- ▶ Erstellung von Layouts erinnert an HTML-Code
- ▶ Jedes Layout darf nur ein Wurzelement enthalten (View, ViewGroup, Merge)
- ▶ Darunter beliebig verschachtelte Layout-Hierarchie
- ▶ Speichern von Layouts unter *res/layouts/*
- ▶ Zugriff im Quellcode über automatisch generierte Klasse *R (gen/)*
- ▶ Zugriff in anderen Layouts über *@[package:]layout/filename*

## Allgemeines

- ▶ Struktur-Elemente, die andere Views anordnen
- ▶ Bsp.: LinearLayout, FrameLayout oder RelativeLayout
- ▶ Weitere ViewGroups ohne Kindelemente (Bsp: AdapterView)
- ▶ Basis-Attribute ID (optional), sowie Breite und Höhe

## Android-IDs

Eindeutige ID eines Layout-Elements wird über *android:id* in der Form *@+id/name* deklariert und kann mit *R.id.name* referenziert werden. + signalisiert dabei, dass es sich um eine neue ID handelt.

Alternativ kann eine ID als Resource deklariert und mit *@id/name* referenziert werden. Die XML-Deklaration kann unter einem beliebigen Dateinamen in */res/values* abgelegt werden.

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

Listing : Deklaration von IDs

## Bemaßungen

- ▶ Bemaßungen eines ViewGroups mit den Attributen *android:layout\_width* und *android:layout\_height* entweder explizit oder implizit
- ▶ Explizite Angabe erfolgt dabei als Wert (px, dp, sp, pt, in, mm) oder als Resource
- ▶ Implizite über die Schlüsselwörter *fill\_parent*, *match\_parent* oder *wrap\_content*

Schlüsselwort	Beschreibung
<i>match_parent</i>	Weist aktuellem Objekt die Größe des Eltern-Elements zu (löst <i>fill_parent</i> ab)
<i>fill_parent</i>	Weist aktuellen Objekt die Größe des Eltern-Elements zu
<i>wrap_content</i>	Sorgt für Anpassung der Größe, sodass Inhalt umschlossen wird

Alternativ kann man auch eine Resource im Quellcode über die Klasse *R* oder in einer XMI-Datei mit dem Schlüssel *@[package:]dimen/dimension\_name* referenzieren.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="imagebox_height">64dp</dimen>
    <dimen name="imagebox_width">32dp</dimen>
5 </resources>
```

### Listing : Deklaration von Dimensionen



## Allgemeines

- ▶ Allein stehendes Element einer grafischen Oberfläche
- ▶ Nimmt rechteckige Fläche des Bildschirms ein
- ▶ Kümmt sich um die Ausgabe und das Verarbeiten von Ereignissen
- ▶ Textfelder, Buttons und Eingabefelder, wie beispielsweise *TextView*
- ▶ Deklaration per XML oder im Quellcode
- ▶ Verwaltung der Basis-Attribute, wie bei ViewGroups

## RequestFocus

- ▶ Gibt View bzw. ViewGroup direkt bei Initialisierung den Fokus
- ▶ XML-Deklaration als leeres Element `<requestFocus>`
- ▶ Zuweisung eines solchen Elements nur einmal pro Datei

## Include

- ▶ Verbindet bereits existierende Layouts
- ▶ Element *<include>*
- ▶ Einziges eigenes Attribut – Name des einzubinden Layouts
- ▶ Optional eine ID, sowie Breite und Höhe des Layouts
- ▶ Werte überschreiben die Werte des Wurzelknotens im eingebundenen Layout
- ▶ Änderungen an den Bemaßungen mit *android:layout\_width* und *android:layout\_height* nur für beide Attribute möglich

### ViewStub

Alternativ kann anstatt des *<include>*-Elements auch ein *ViewStub* verwendet werden.

## Merge

- ▶ Wurzelement um Layouts zu verbinden
- ▶ Sinnvoll wenn Layouts mit *<include>* verbunden werden sollen
- ▶ Flacherer und dadurch einfacherer Aufbau der View-Hierarchie

# View-Klassen

# Contents

- 7 TextView
- 8 EditText
- 9 Button
- 10 CompoundButton
- 11 CheckBox
- 12 ToggleButton
- 13 RadioButton
- 14 RadioGroup
- 15 CheckedTextView
- 16 ImageView
- 17 ImageButton
- 18 ProgressBar
- 19 Spinner
- 20 Chronometer
- 21 WebView
- 22 TimePicker
- 23 DatePicker

## Allgemeines

- ▶ Ausgabe von Text auf dem Bildschirm
- ▶ Formatierung über Attribute

Attribut	Beschreibung
android:text	Der anzuzeigene Text
android:textSize	Schriftgröße [dimension]
android:textColor	Farbe des Textes [color]
android:textStyle	Stil des Texts (bold, italic, bolditalic) [flag]
android:typeface	Schriftart (normal, sans, serif, monospace) in der der Text angezeigt werden soll [enum]
android:textAllCaps	Flag um Text in Großbuchstaben auszugeben [boolean]
android:shadowColor	Farbe des Text-Schattens [color]
android:shadowRadius	Radius des Text-Schattens [double]
android:shadowDx	Verschiebung des Schattens in x-Richtung [double]
android:shadowDy	Verschiebung des Schattens in y-Richtung [double]

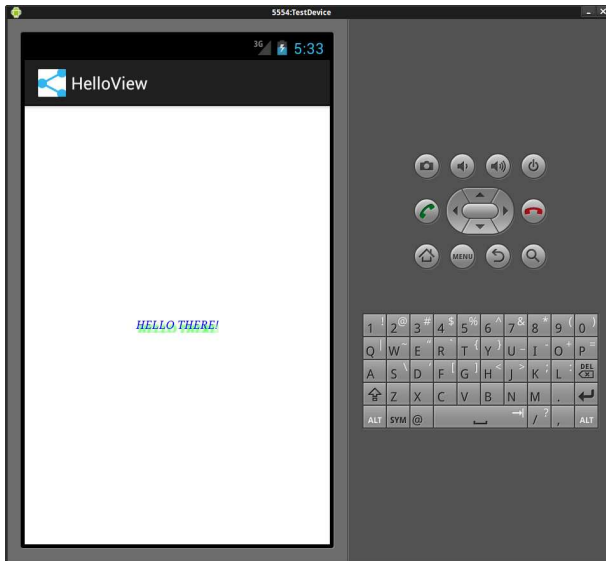
## Deklaration im Layout

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
5  android:text="Hello There!"
  android:textSize="12dp"
  android:textColor="#0000ff"
  android:textStyle="italic"
  android:typeface="serif"
  android:textAllCaps="true"
10 android:shadowColor="#00ff00"
  android:shadowRadius="2.0"
  android:shadowDx="3.0"
  android:shadowDy="7.0" />
```

Listing : TextView



# Screenshot



## Allgemeines

- ▶ Abgeleitet von der Klasse `TextView`
- ▶ Erlaubt Editieren von Text
- ▶ Formatierungen, wie bei `TextView` möglich

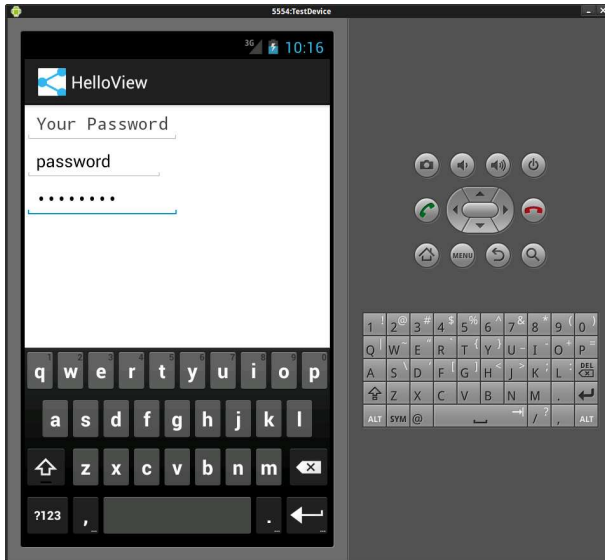
Attribut	Beschreibung
<code>android:hint</code>	Text der angezeigt werden soll, wenn das Eingabefeld leer ist
<code>android:lines</code>	Höhe des Eingabefeldes in Zeilen
<code>android:maxLength</code>	Maximale Länge der Eingabe
<code>android:password</code>	Erlaubt das verstecken des eingegeben Texts und deaktiviert die Autovervollständigung [boolean]
<code>android:textColorHint</code>	Farbe des Standard-Textes [color]

## Deklaration im Layout

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Your Password"
5   android:textColorHint="#454545"
    android:password="true"
    android:maxLength="10" />
```

Listing : EditText

# Screenshot



## Allgemeines

- ▶ Abgeleitet von der Klasse `TextView`
- ▶ Erzeugt anklickbare Schaltfläche
- ▶ Erlaubt direkte Verlinkung mit Callback-Methode
- ▶ Alternativ Verwendung von *`OnClickListener`*

Attribut	Beschreibung
<code>android:onClick</code>	Definiert eine Methode der Aktivität, die beim klicken auf den Button ausgeführt werden soll

## Deklaration & Implementierung

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="KlickMe"
5    android:onClick="klickMeClicked" />
```

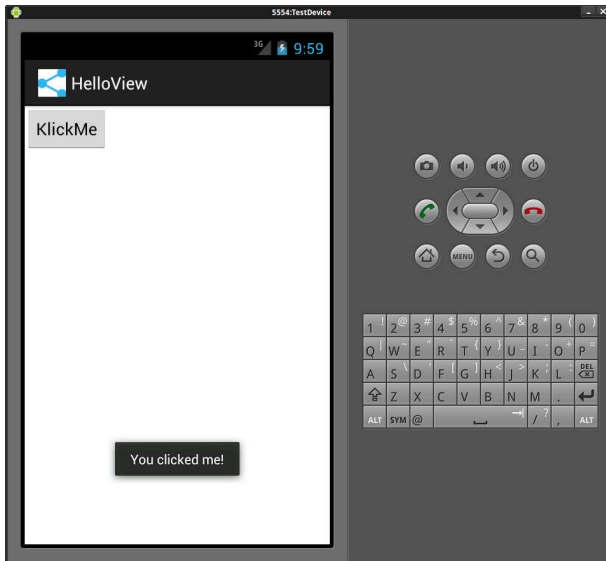
Listing : Button Layout

```
public class HelloView extends Activity {
    ...

    public void onClickMeClicked(View view) {
5        Toast.makeText(this, "You clicked me!", Toast.LENGTH_LONG).show();
    }
}
```

Listing : Button OnClickListener

# Screenshot



## Allgemeines

- ▶ Abstrakte View-Klasse  
→ Kann nicht direkt verwendet werden
- ▶ Abgeleitet von der Klasse Button
- ▶ Kennt zwei Zustände aktiviert (*checked*) und deaktiviert (*unchecked*)
- ▶ Basis-Klasse für z.B. CheckBox, RadioButton und ToggleButton

Attribut	Beschreibung
android:button	Legt ein Bild fest, welches auf der Schaltfläche angezeigt werden soll [drawable]
android:checked	Gibt den initialen Zustand der Schaltfläche an [boolean]

```
<CompoundButton
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="Toogle Me!"
5    android:button="@drawable/ic_launcher"
    android:checked="true" />
```

Listing : CompoundButton



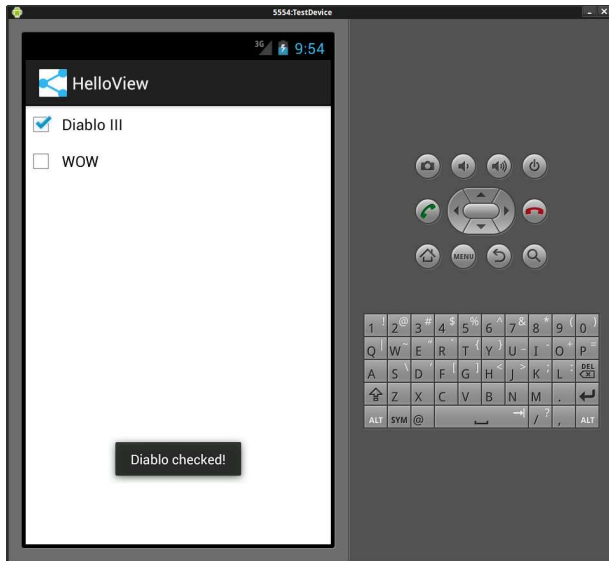
## Allgemeines

- ▶ Abgeleitet von der Klasse CompoundButton
- ▶ Kennt zwei Zustände aktiviert (*checked*) und deaktiviert (*unchecked*)

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
5  android:layout_height="fill_parent">
  <CheckBox
    android:id="@+id/cbxDiablo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
10  android:text="Diablo III"
    android:onClick="onDiabloChecked" />
  <CheckBox
    android:id="@+id/cbxWow"
    android:layout_width="wrap_content"
15  android:layout_height="wrap_content"
    android:text="WOW"
    android:onClick="onWowChecked" />
</LinearLayout>
```

Listing : CheckBox

# Screenshot



## Callback Implementierung

- ▶ Verwendung mehrerer zusammengehöriger Checkboxes möglich
- ▶ Zuweisung eines Callbacks über *android:onClick*-Attribut
- ▶ Nutzung des gleichen Callbacks möglich

```
public class HelloView extends Activity {  
    ...  
  
    public void onGameCbxClicked(View view) {  
5      // Get the checkbox state ...  
        boolean checked = ((CheckBox) view).isChecked();  
  
        // ... and check which checkbox was clicked.  
        switch(view.getId()) {  
10         case R.id.cbxDiablo:  
            Toast.makeText(this, "Diablo checked!", Toast.LENGTH_LONG).show();  
            break;  
         case R.id.cbxWow:  
15         Toast.makeText(this, "WOW checked!", Toast.LENGTH_LONG).show();  
            break;  
        }  
    }  
}
```

Listing : Methode für mehrere CheckBoxen

## Allgemeines

- ▶ Abgeleitet von der Klasse CompoundButton
- ▶ Kennt zwei Zustände aktiviert (*checked*) und deaktiviert (*unchecked*)
- ▶ Dient als An- und Aus-Schalter
- ▶ Weißt jedem Zustand einen Namen zu

```
<ToggleButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Sound on"
5   android:textOff="Sound off"
    android:onClick="onToggled" />
```

Listing : ToggleButton

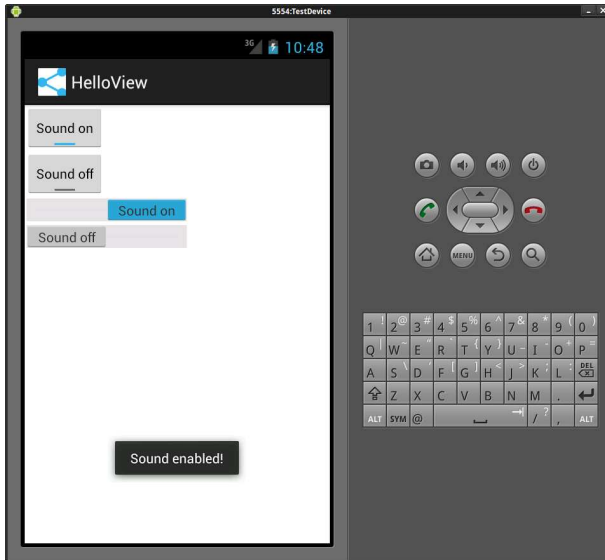
## Alternative: Slider

- ▶ Eingeführt in Android 4.0
- ▶ Abgeleitet von der Klasse CompoundButton
- ▶ Verhalten wie ToggleButton
- ▶ Ausgabe als Slider

```
<Switch
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Sound on"
5    android:textOff="Sound off" />
```

Listing : Switch

# Screenshot



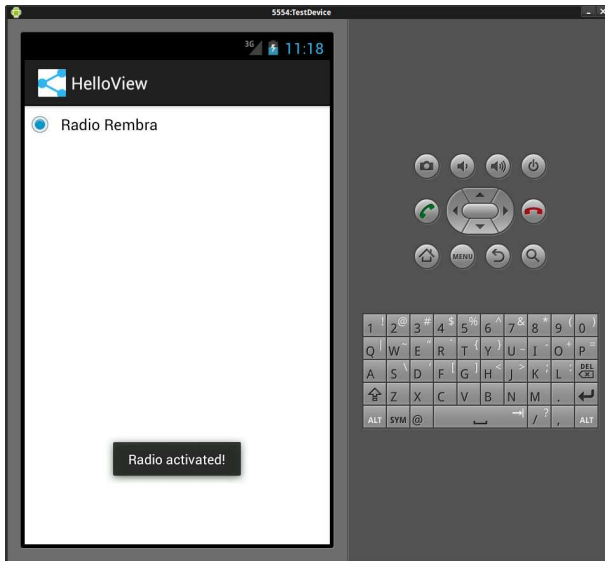
## Allgemeines

- ▶ Abgeleitet von der Klasse CompoundButton
- ▶ Einmal aktiviert immer aktiviert
- ▶ Gruppierung in RadioGroup sinnvoll

```
<RadioButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Radio Rembra"  
5    android:onClick="onToggled"/>
```

Listing : RadioButton

# Screenshot





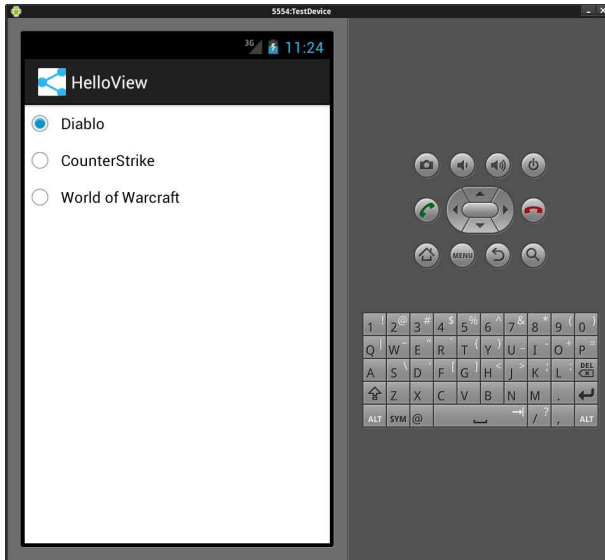
## Allgemeines

- ▶ Abgeleitet von der Klasse `LinearLayout`
- ▶ Dienen der Gruppierung von `RadioButtons`
- ▶ Immer nur ein einzelner `RadioButton` aktiviert

```
<RadioGroup
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
5  android:layout_height="match_parent" >
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Diablo"
10  android:onClick="onRadio" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CounterStrike"
15  android:onClick="onRadio" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="World of Warcraft"
20  android:onClick="onRadio" />
</RadioGroup>
```

Listing : RadioGroup

# Screenshot



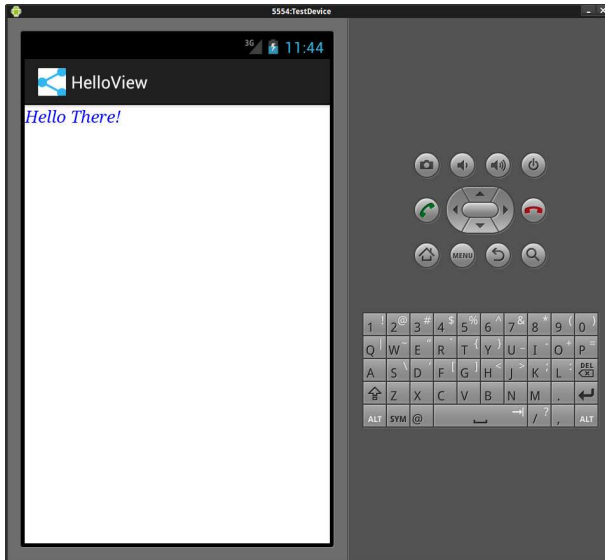
## Allgemeines

- ▶ Abgeleitet von TextView
- ▶ Implementiert das Checkable Interface
- ▶ Kennt Zustände aktiviert (*checked*) und deaktiviert (*unchecked*)
- ▶ Einsatz sinnvoll mit Spinnern und ListViews, wenn *setChoiceMode()* auf *CHOICE\_MODE\_NONE* gesetzt

```
<CheckedTextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello There!"
5   android:textSize="17dp"
    android:textColor="#0000ff"
    android:textStyle="italic"
    android:typeface="serif"
    android:onClick="onTextChecked" />
```

Listing : CheckedTextView

# Screenshot



## Allgemeines

- ▶ Anzeige eines Bildes
- ▶ Kümmert sich um die Bemaßung des Bildes
- ▶ Formatierungen (Bsp. Einfärbung) über Attribute möglich

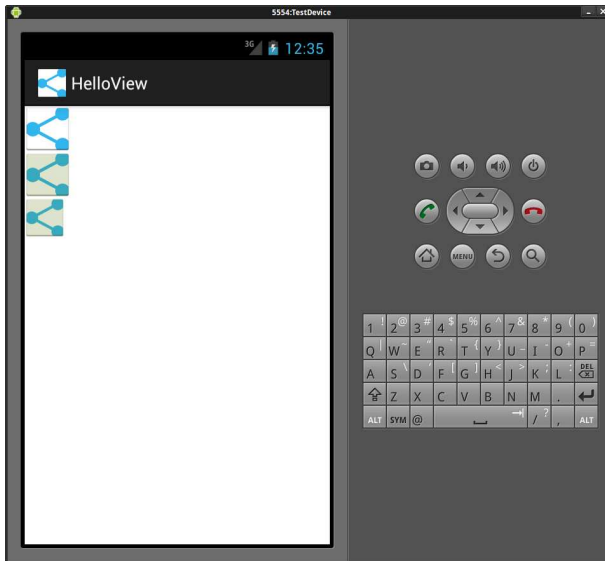
Attribut	Beschreibung
android:src	Die zu verwendende Bilddatei [drawable]
android:tint	Farbe für Bildeinfärbung [color]
android:adjustViewBounds	Legt fest, ob das ImageView seine Bemaßung an die seines Bildes anpasst [boolean]
android:scaleType	Legt fest, wie das Bild skaliert und bewegt werden soll um die erlaubten Bemaßungen einzuhalten [enum]
android:cropToPadding	Legt fest ob ein Bild zugeschnitten werden soll um in seinen Bereich zu passen [boolean]
android:baselineAlignBottom	Legt fest ob das Bild an der Grundlinie basierend auf der unteren kante ausgerichtet wird [boolean]
android:baseline	Legt die Grundlinie des Bildes fest [flag]
android:maxHeight	Maximale Höhe [dimension]
android:maxWidth	Maximale Breite [dimension]

## Deklaration im Layout

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
5    android:maxHeight="7dp"
    android:maxLength="7dp"
    android:tint="#345678" />
```

Listing : ImageView

# Screenshot



## Allgemeines

- ▶ Abgeleitet von `ImageView`
- ▶ Sieht normalerweise aus wie Button mit Bild
- ▶ Angabe von Callback mit *android:onClick*

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
5    android:onClick="onImageButtonClicked" />
```

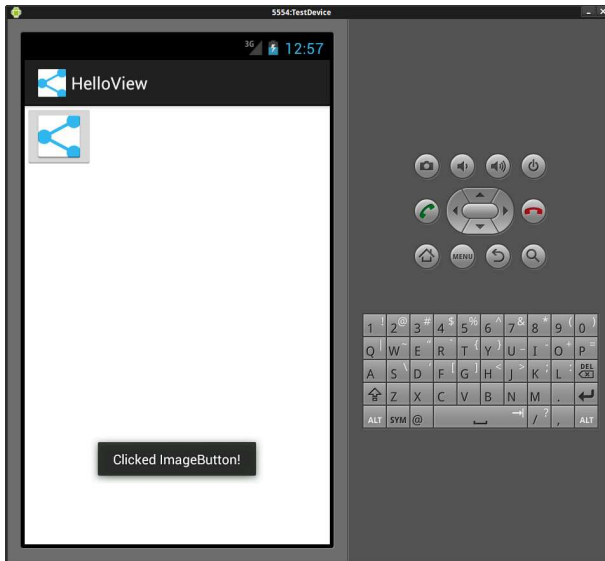
Listing : ImageButton

```
public void onImageButtonClicked(View view) {
    Toast.makeText(this, "Clicked ImageButton!", Toast.LENGTH_LONG).show();
}
```

Listing : Methode onImageButtonClicked



# Screenshot



## Anmerkung

### Event-Hintergründe

Es ist möglich die Änderung der Hintergrundfarbe bei verschiedenen Benutzeraktionen, wie beispielsweise einem Klick, selbst festzulegen. Zu diesem Zweck kann man in XML einen *selector* definieren, der anhand der verschiedenen Button-Zustände ein Bild auswählt.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true"
        android:drawable="@drawable/button_pressed" />
5   <item android:state_focused="true"
        android:drawable="@drawable/button_focused" />
  <item android:drawable="@drawable/button_normal" />
</selector>
```

Listing : Deklaration von Button-Farben

Diese Datei legt man wie ein normales Bild unter *res/drawable/* ab.

## Allgemeines

- ▶ Visualisierung des Fortschritts eines laufenden Prozesses
- ▶ Unterscheidung in nicht deterministisch und deterministisch
- ▶ Möglichkeit einen zweiten Ladebalken anzuzeigen (Bsp. Buffering)

Android bietet verschiedene Styles, die über das Attribut *style* in der Form *@android:style/style\_name* ausgewählt werden:

- ▶ `Widget.ProgressBar.Inverse`
- ▶ `Widget.ProgressBar.Horizontal`
- ▶ `Widget.ProgressBar.Small`
- ▶ `Widget.ProgressBar.Small.Inverse`
- ▶ `Widget.ProgressBar.Large`
- ▶ `Widget.ProgressBar.Large.Inverse`

# Attribute

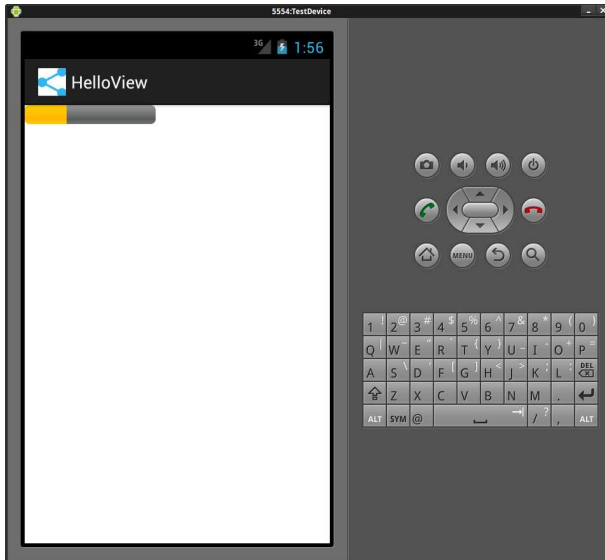
Attribut	Beschreibung
android:indeterminate	Aktiviert den <i>nicht deterministischen</i> Modus einer ProgressBar
android:max	Legt den maximalen Wert fest [double]
android:maxHeight	Maximale Höhe der Bar [dimension]
android:maxLength	Maximale Breite der Bar [dimension]
android:minHeight	Minimale Höhe der Bar [dimension]
android:minWidth	Minimale Breite der Bar [dimension]
android:progress	Der Wert des aktuellen Fortschritts (zwischen 0 und max)
android:progressDrawable	Bild die ProgressBar
android:secondaryProgress	Der Wert des aktuellen sekundären Fortschritts (zwischen 0 und max)

## Deklaration im Layout

```
<ProgressBar
    android:id="@+id/progress_bar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
5    android:minWidth="137dp"
    android:max="50"
    style="@android:style/Widget.ProgressBar.Horizontal" />
```

Listing : ProgressBar

# Screenshot



# Implementierung

```
public class HelloView extends Activity {  
    private ProgressBar mProgress;  
  
    @Override  
5    public void onCreate(Bundle savedInstanceState) {  
        ...  
  
        mProgress = (ProgressBar) findViewById(R.id.progress_bar);  
        new Thread(new Runnable() {  
10            public void run() {  
                for(int stat = 0; stat < 100; stat++) {  
                    try {  
                        Thread.sleep(1000);  
                        mProgress.setProgress(stat);  
15                    } catch (InterruptedException e) {  
                        e.printStackTrace();  
                    }  
                }  
            }  
20        }).start();  
    }  
    ...  
}
```

Listing : Aktualisierung einer ProgressBar

## Allgemeines

- ▶ Stellen ein DropDown-Menü bereit
- ▶ Alternativ kann Dialog angezeigt werden
- ▶ Platz sparende Alternative zu RadioGroups
- ▶ Initialisierung mit einem Start-Wert möglich

Attribut	Beschreibung
android:dropDownWidth	Breite des DropDowns für Spinner im DropDown-Modus
android:popupBackground	Hintergrundbild für Spinner im DropDown-Modus
android:prompt	Titel des Spinner Dialogs [resource]
android:spinnerMode	Anzeige Modus eines Spinners (dialog, dropdown) [int]

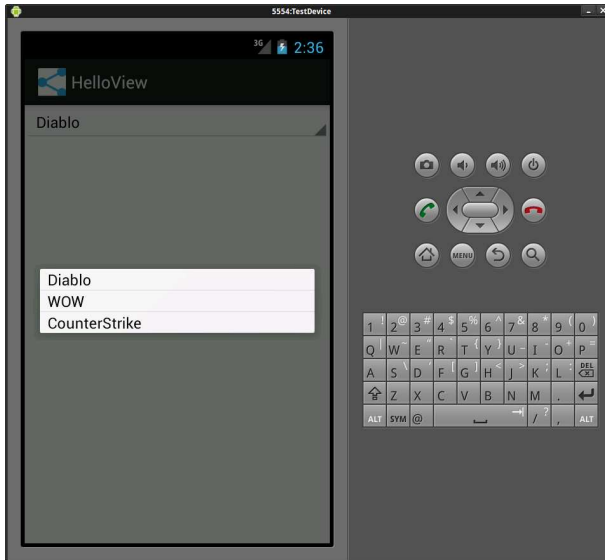


## Deklaration & Implementierung

```
public class HelloView extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        ...  
5        Spinner spinner = (Spinner) findViewById(R.id.spinner);  
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(  
            this, R.array.games_array, android.R.layout.simple_spinner_item);  
        spinner.setAdapter(adapter);  
10    }  
    ...  
}
```

Listing : Ein Spinner-Adapter

# Screenshot



## Anmerkung

### String-Arrays

Auswahllisten, wie Spinner, können mit String-Array-Ressourcen befüllt und sprachunabhängig gestaltet werden. Deklaration in der Sprachdatei *string.xml* unter */res/values*.

```
<resources>
    <string name="app_name">Hello Views</string>
    ...
5    <string-array name="games_array">
        <item>Diablo</item>
        <item>WoW</item>
        <item>CounterStrike</item>
10   </string-array>
</resources>
```

Listing : Deklaration eines String-Arrays

## Allgemeines

- ▶ Benutzeroberfläche für Timer
- ▶ Abgeleitet von Klasse *TextView*
- ▶ Startwert mit *elapsedRealtime()* Methode aus *SystemClock* Klasse setzen
- ▶ Alternativ: Automatisch mit *start()* initialisieren
- ▶ Format *MM:SS* oder *H:MM:SS*
- ▶ Format kann mit *setFormat(String)* geändert werden

Attribut	Beschreibung
android:format	String der ein Ausgabeformat festlegt. Das erste Vorkommen von %s wird durch den aktuellen wert des Timers im Format <i>MM:SS</i> oder <i>H:MM:SS</i> ersetzt.

## Deklaration im Layout

```
5  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Chronometer
        android:id="@+id/chronometer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:format="Relapsed time since start: %s" />
10  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/startStopwatch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start"
        android:onClick="onStartStop" />
20  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Reset"
    android:onClick="onReset" />
25  </LinearLayout>
</LinearLayout>
```

Listing : Chronometer

# Implementierung

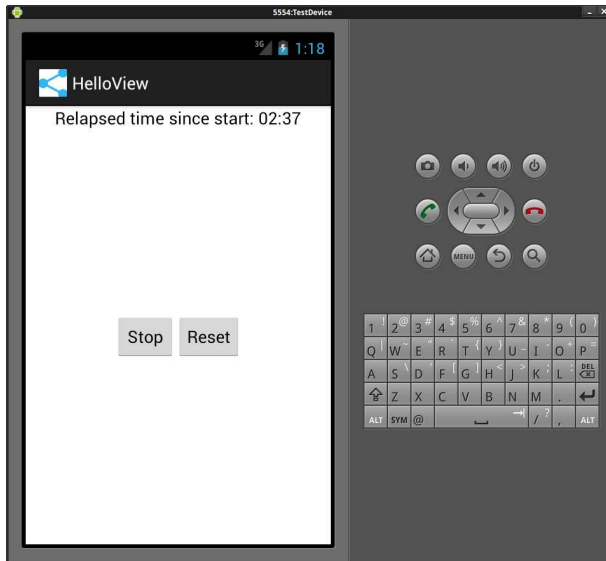
```
public class HelloView extends Activity {
    private Chronometer chron;
    private Button startStop;
    private boolean isChronRunnig = false;
5    private long elapsed = 0;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
10        setContentView(R.layout.activity_hello_view);
        chron = (Chronometer) this.findViewById(R.id.chronometer);
        startStop = (Button) this.findViewById(R.id.startStopwatch);
    }

15    public void onStartStop(View v) {
        if (!isChronRunnig) {
            chron.setBase(SystemClock.elapsedRealtime() - elapsed);
            isChronRunnig = true;
            startStop.setText("Stop");
20            chron.start();
        } else {
            elapsed = SystemClock.elapsedRealtime() - chron.getBase();
            isChronRunnig = false;
            startStop.setText("Start");
25            chron.stop();
        }
    }

    public void onReset(View v) {
30        elapsed = 0;
        isChronRunnig = false;
        chron.stop();
        chron.setText("00:00");
        startStop.setText("Start");
35    }
}
```

# Screenshot



## Allgemeines

- ▶ Ermöglicht Anzeige von Webseiten innerhalb einer Applikation
- ▶ Abgeleitet von der Klasse *ViewGroups* oder genauer von *AbsolutLayout*
- ▶ Anzeigen der Webinhalte übernimmt die *WebKit* Engine
- ▶ Unterstützt nicht von Haus aus Web typische Besonderheiten, wie JavaScript
- ▶ Inhalte mit *loadUrl()* oder HTML-Code mit *loadData()* laden

### Zugriffsrechte für das Internet

Um einer Applikation den Zugriff auf das Internet zu ermöglichen, benötigt sie die entsprechenden Rechte. Das bedeutet, dass in der Android Manifest-Datei diese Rechte gesetzt werden müssen. Dazu muss die Zeile aus Listing eingebunden werden.

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

Listing : Internet Zugriffsrechte



## Web 2.0 Funktionen

### JavaScript einschalten

Auch wenn ein `WebView` nicht standardmäßig JavaScript unterstützt kann diese Funktionalität aktiviert werden. Zu diesem Zweck verwaltet jedes `WebView` seine Einstellungen in einem `WebSettings` Objekt, auf das mit der Methode `getSettings()` zugegriffen werden kann. Mit einem Aufruf der Methode `setJavaScriptEnabled(true)` der Klasse `WebSettings` kann die Unterstützung von JavaScript für das entsprechende `WebView` aktiviert werden.

### HTML-Standard

Man muss bei der Verwendung von `WebView` immer darauf achten, dass es nicht nur Probleme mit komplexeren Webseiten hat, die JavaScript oder Flash einbinden, sondern dass es auch Einschränkungen in Bezug auf den HTML-Code macht.

## Funktionalitäten erweitern

**WebChromeClient** Kümmt sich um die Verarbeitung von Events, die die grafische Oberfläche beeinflussen.

**WebViewClient** Kümmt sich um die Verarbeitung von Events, die die Anzeige des Inhalts beeinflussen.

**WebSettings** Änderungen an den Einstellungen, wie Aktivierung von JavaScript.

### Web-Applikationen

Mit WebView ist es möglich bereits existierende Web-Applikationen direkt in Android zu integrieren.

Dazu wird es ermöglicht Teile des oftmals in Web-Applikationen verwendeten JavaScript-Codes an Android-Code zu binden. So ist es beispielsweise möglich alle Ausgaben, die durch ein JavaScript-Alert erzeugt würden in Android-Toasts ausgeben zu lassen. Android ermöglicht dies durch die Implementierung von *JavaScriptInterfaces*.

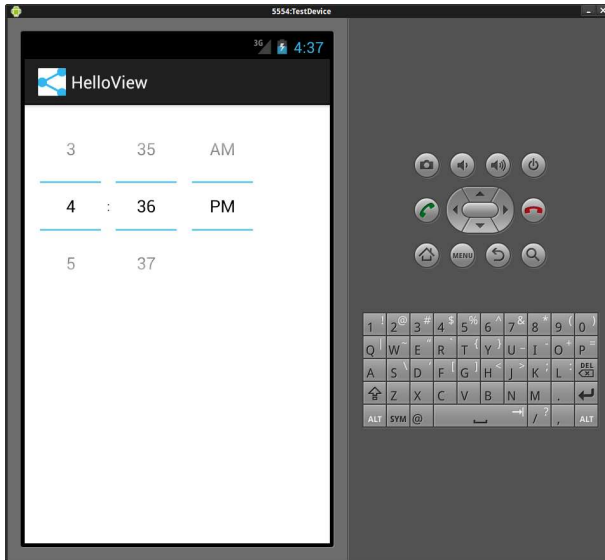
## Allgemeines

- ▶ *TimePicker* ist abgeleitet von Klasse *FrameLayout*
- ▶ Oberfläche zur Auswahl einer Uhrzeit im AM/PM-Format

```
<TimePicker  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

Listing : TimePicker

# Screenshot



## Allgemeines

- ▶ *DatePicker* ist abgeleitet von Klasse *FrameLayout*
- ▶ Oberfläche zur Auswahl eines Datums
- ▶ Kann als Spinner oder Kalender (*CalendarView*) angezeigt werden

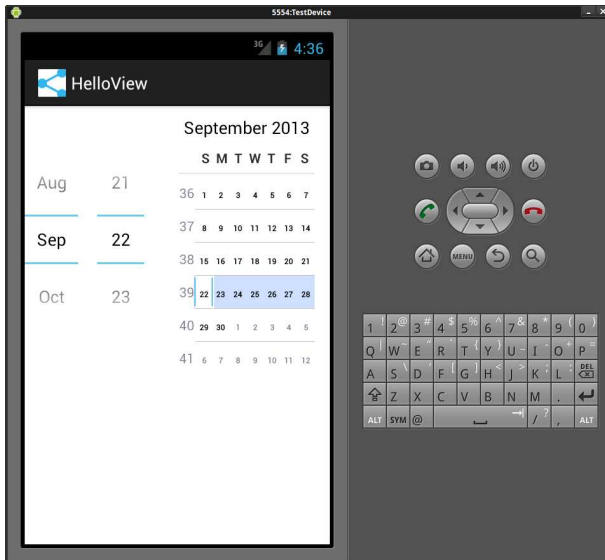
Attribut	Beschreibung
android:startYear	Das erste Jahr, das als Eingabe möglich sein soll [int]
android:endYear	Das letzte Jahr, das als Eingabe möglich sein soll [int]
android:maxDate	Das minimal angezeigte Datum im CalendarView (Format: mm/dd/yyyy) [string]
android:minDate	Das maximal angezeigte Datum im CalendarView (Format: mm/dd/yyyy) [string]
android:spinnersShown	Gibt an ob Spinner angezeigt werden sollen [boolean]
android:calendarViewShown	Gibt an ob ein CalendarView angezeigt werden soll [boolean]

## Deklaration im Layout

```
<DatePicker
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:endYear="2030"
5   android:minDate="11/01/2012"
    android:calendarViewShown="true" />
```

Listing : DatePicker

# Screenshot



# ViewGroup-Klassen



# Contents

24 LinearLayout

25 RelativeLayout

26 FrameLayout

27 TableLayout

28 ScrollView

## Allgemeines

- ▶ Ermöglicht eine horizontale oder vertikale Anordnung seiner Kindelemente
- ▶ Reihenfolge in der Definition der Kindelemente wird eingehalten
- ▶ Unterstützt Gewichtungen von Kindelementen (*android:layout\_weight*)

Attribut	Beschreibung
android:divider	Definiert ein Drawable, das als Trenner zwischen vertikal angeordneten Elementen angezeigt wird [drawable]
android:gravity	Definiert, wie der Inhalt des Layouts angeordnet werden soll [int]
android:measureWithLargestChild	Falls dieses Flag aktiviert wird, werden alle Kindelemente mit Gewichtung dazu gezwungen ihre minimale Größe an die des größten Elements anzupassen [boolean]
android:orientation	Legt fest, ob die Kindelemente horizontal oder vertikal angeordnet werden [int]
android:weightSum	Legt die maximale Summe der Gewichtungen fest

# Gewichtung

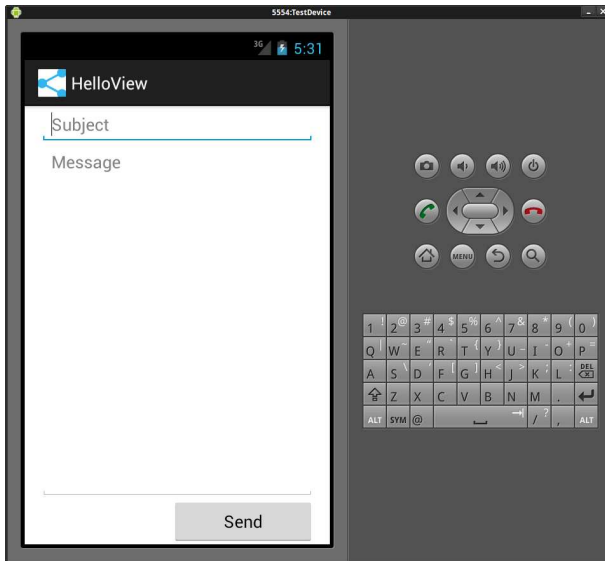
## Layouts gewichten

Um allen Kindelementen eines LinearLayouts den gleichen Raum zu geben kann die Breite des Kindelements im horizontalen Layout auf *0dp*, im vertikalen Layout die Höhe auf *0dp* gesetzt werden. Damit dies jedoch funktioniert müssen alle Kindelement mit 1 gewichtet werden. Dazu setzt man das Attribut *android:layout\_weight* jedes Kindelements auf 1.

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
5   <EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="Subject" />
10  <EditText
      android:layout_width="fill_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="Message" />
15  <Button
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:text="Send" />
20 </LinearLayout>
```

Listing : LinearLayout

# Screenshot



## Allgemeines

- ▶ Ordnet seine Kindelemente relativ zueinander an
- ▶ View kann seine Position relativ zu anderen Views auf der selben Ebene oder zum Vater-Element angeben kann

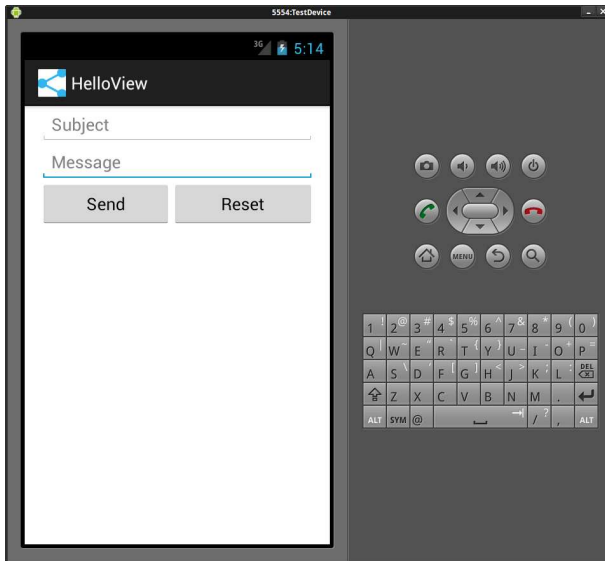
Attribut	Beschreibung
android:gravity	Positionierung der Inhalte innerhalb des Views [int]
android:layout_alignParentTop	Legt die Oberkante des Kindelements an die Oberkante des Vaters [boolean]
android:layout_centerInParent	Vertikale & horizontale Zentrierung im Vater[boolean]
android:layout_centerHorizontal	Horizontale Zentrierung im Vater[boolean]
android:layout_centerVertical	Vertikale Zentrierung im Vater[boolean]
android:layout_above	Positionierung über dem View [resource]
android:layout_below	Positionierung unter dem View [resource]
android:layout_toRightOf	Positionierung rechts neben dem View [resource]
android:layout_toLeftOf	Positionierung links neben dem View [resource]

## Deklaration im Layout

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
5   <Button
      android:id="@+id/reset"
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
10     android:layout_below="@+id/message"
      android:text="Reset" />
   <EditText
      android:id="@+id/message"
      android:layout_width="fill_parent"
15     android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      android:layout_below="@+id/subject"
      android:hint="Message"
      android:gravity="top" />
20   <Button
      android:id="@+id/send"
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:layout_below="@+id/message"
25     android:layout_toLeftOf="@+id/reset"
      android:text="Send" />
   <EditText
      android:id="@+id/subject"
      android:layout_width="fill_parent"
30     android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:hint="Subject" />
</RelativeLayout>
```

Listing : RelativeLayout

# Screenshot



## Allgemeines

- ▶ Hält einen gewissen Bereich auf dem Bildschirm für ein einzelnes View frei
- ▶ Mehrere Kindelemente in einem FrameLayout können sich überlappen
- ▶ Größe des FrameLayouts wird durch größtes Kindelement festgelegt (zzgl. Padding)
- ▶ Egal ob Kinder sichtbar oder nicht

Attribut	Beschreibung
android:foreground	Legt ein Bild fest, dass über dem Inhalt gezeichnet werden soll [drawable]
android:foregroundGravity	Legt die Ausrichtung des Bildes fest [int]
android:measureAllChildren	Legt fest ob die Bemaßung aller Kindelemente angepasst werden soll [boolean]

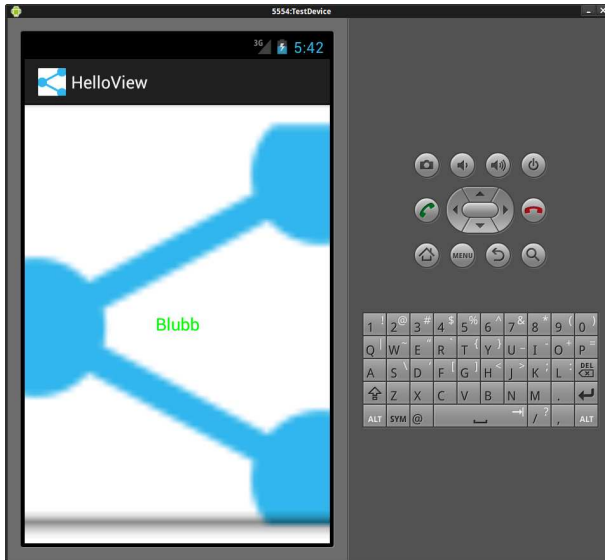


## Deklaration im Layout

```
<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
5   <ImageView
        android:src="@drawable/ic_launcher"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:scaleType="centerCrop" />
10  <TextView
        android:text="Blubb"
        android:textColor="#00ff00"
        android:textSize="18dp"
        android:layout_height="fill_parent"
15  android:layout_width="fill_parent"
        android:gravity="center" />
</FrameLayout>
```

Listing : FrameLayout

# Screenshot



## Allgemeines

- ▶ Platziert Inhalte in einer Tabellenstruktur
- ▶ Zeilen werden explizit angegeben, Spalten implizit angenommen werden

Attribut	Beschreibung
android:collapseColumns	Index der Spalten die zusammengefügt werden sollen (beginnend bei Null) [int,boolean]
android:shrinkColumns	Spalten werden automatisch verkleinert [boolean]
android:stretchColumns	Spalten werden automatisch vergrößert [boolean]
android:layout_column	Spalte in der ein View eingefügt werden soll [int]
android:layout_span	Anzahl an Spalten die ein View einnehmen soll [int]

## Deklaration im Layout

```
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
5  android:shrinkColumns="*"
  android:stretchColumns="*" >
  <TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
10    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Test Mailer"
      android:layout_span="4" />
15    </TableRow>
    <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
20      <TextView
        android:text="Message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
      <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Type your message here ..."
        android:layout_span="3" />
25      </TableRow>
30    ...
```

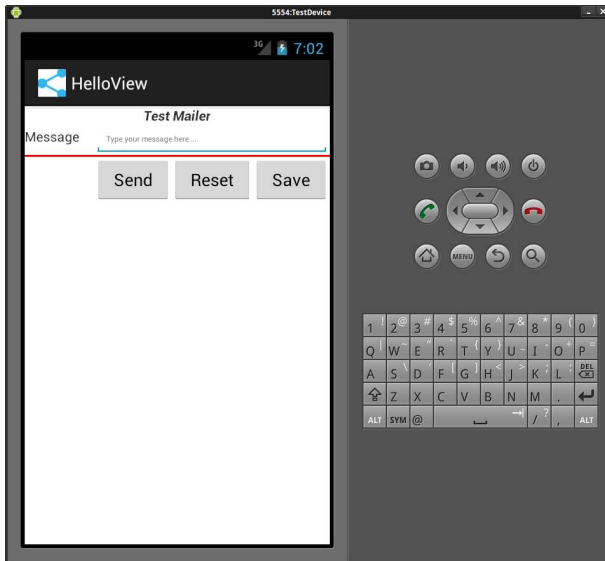
Listing : TableLayout

## Deklaration im Layout II

```
...  
  
    <!-- just draw a red line -->  
    <View  
5      android:layout_height="2dip"  
      android:layout_width="match_parent"  
      android:background="#FF0000" />  
  
    <TableRow  
10     android:layout_width="match_parent"  
      android:layout_height="wrap_content">  
      <Button  
      android:text="Send"  
      android:layout_column="1"  
15     android:layout_width="wrap_content"  
      android:layout_height="wrap_content" />  
      <Button  
      android:text="Reset"  
      android:layout_width="wrap_content"  
20     android:layout_height="wrap_content" />  
      <Button  
      android:text="Save"  
      android:layout_width="wrap_content"  
      android:layout_height="wrap_content" />  
25    </TableRow>  
  </TableLayout>
```

Listing : TableLayout

# Screenshot



## Allgemeines

- ▶ Abgeleitet von `FrameLayout`
- ▶ Ermöglicht, dass das Layout größer ist als das eigentliche Display erlaubt
- ▶ Kann nur ein Kind enthalten (sichtbar oder nicht)

### ListViews

Ein `ScrollView` sollte niemals mit einem `ListView` verwendet werden, da sich dies bereits selbst um das Scrollen seiner Elemente kümmert. Sollte man dies dennoch tun, zwingt man das `ListView` dazu sich auf die benötigte Länge auszudehnen um alle Elemente anzeigen zu können. Die in `ListView` implementierten Optimierungen zur Anzeige großer Listen würden somit umgangen werden.

### Horizontales Scrollen

Ein `ScrollView` kümmert sich nur um das vertikale Scrollen seines Inhalts. Sollte es einmal nötig sein den Inhalt horizontal zu scrollen, so sollte man auf die Klasse `HorizontalScrollView` zurückgreifen.

## Deklaration im Layout

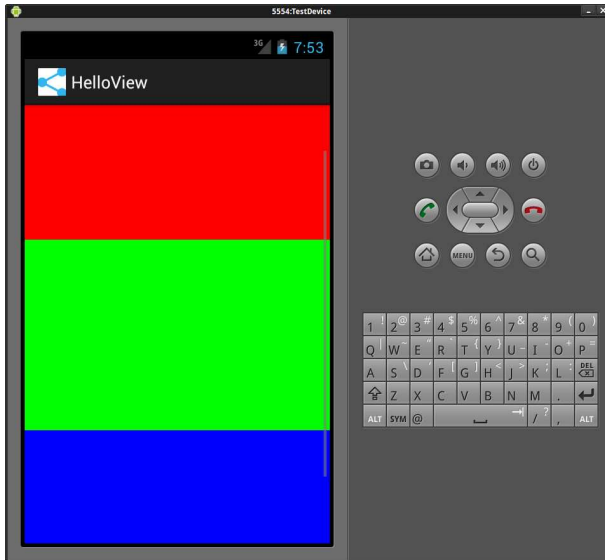
Attribut	Beschreibung
android:fillViewport	Ermöglicht es den Inhalt so auszudehnen, dass er den verfügbaren Display-Bereich nutzt [boolean]

```
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
5   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical" >
      <TextView
10         android:layout_width="match_parent"
           android:layout_height="200dp"
           android:background="#ff0000" />
      <TextView
15         android:layout_width="match_parent"
           android:layout_height="200dp"
           android:background="#00ff00" />
      <TextView
20         android:layout_width="match_parent"
           android:layout_height="200dp"
           android:background="#0000ff" />
    </LinearLayout>
</ScrollView>
```

Listing : ScrollView



# Screenshot



# AdapterView-Klassen

# Contents

29 ListView

30 GridView

31 ExpandableListView

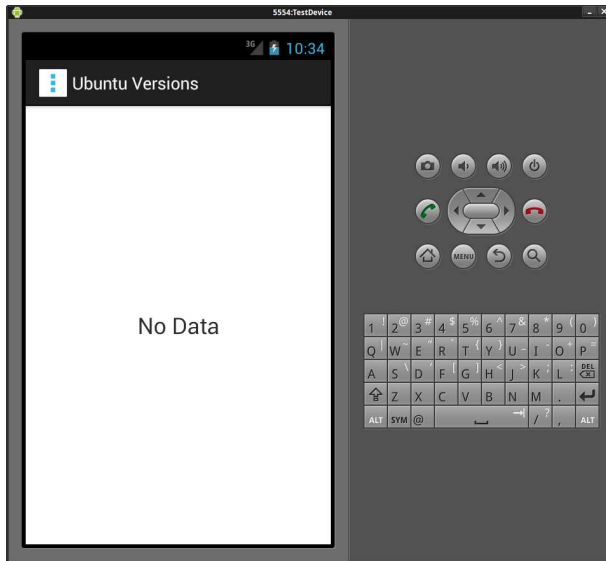
# ListView

- ▶ Scrollbare Liste von Einträgen
- ▶ Nutzt spezielle Aktivität ListActivity
- ▶ ListActivity nutzt automatisch ein Standard-Layout
- ▶ Eigenes Layout kann mit `setContentView()` gesetzt werden
- ▶ Liste muss ID `@android:id/list` tragen
- ▶ View für leere Listen muss ID `@android:id/empty` tragen

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ListView
5        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="27dp" />
10    <TextView
        android:id="@android:id/empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_vertical|center_horizontal"
15        android:textSize="24dp" />
</RelativeLayout>
```

Listing : ListView

# Screenshot



# ListActivity

```
public class HelloList extends ListActivity {
    private ArrayAdapter<CharSequence> adapter;

    @Override
5    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.hello_list);

        // Setup the adapter ...
10       adapter = ArrayAdapter.createFromResource(this, R.array.ubuntu_array,
            android.R.layout.simple_list_item_1);

        // ... and assign it to the listview.
        this.setAdapter(adapter);
15    }

    @Override
    protected void onItemClick(ListView l, View v, int position, long id) {
        // Load the selected item ...
20       String ubuntu = (String) this.adapter.getItem(position);

        // ... and show a short Toast.
        Toast.makeText(this, "Selected " + ubuntu + ".",
            Toast.LENGTH_LONG).show();
25    }
}
```

Listing : Die Klasse HelloList

# Ressourcen

## Array-Ressourcen

Wie Zeichenketten, Bemaßungen und Styles können auch Arrays als Ressourcen hinterlegt werden.

Arrays werden wie gewohnt in XML deklariert. Als Werte des Arrays können praktisch beliebige Resource-Typen, wie Strings, Integer oder Drawables dienen. Dabei können die Typen vermischt werden. Die entstehende Datei kann unter *res/values/arrays.xml* abgelegt werden.

```
<resources>
  <array name="game_icons">
    <item>@drawable/wow</item>
    <item>@drawable/diablo</item>
5    <item>@drawable/cs</item>
  </array>
  <array name="ubuntu_array">
    <item>Ubuntu 4.10 (Warty Warthog)</item>
    <item>Ubuntu 5.04 (Hoary Hedgehog)</item>
10    <item>Ubuntu 5.10 (Breezy Badger)</item>
    ...
  </array>
</resources>
```

Listing : Array Ressourcen

## Ressourcen

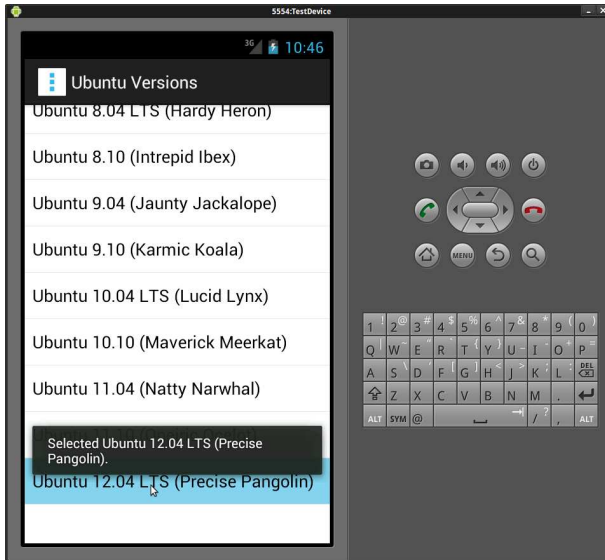
### Android-Layouts

Beim Betrachten der Klasse `HelloList` sollte auffallen, dass ein Layout referenziert wird (`simple_list_item_1`), das bisher nicht erstellt wurde.

Es handelt sich um ein von Android bereitgestelltes Standard-Layout, das mit dem Android-SDK mitgeliefert wird. Es gibt weitere Layouts, wie `alert_dialog`, `date_picker`, `search_bar` und auch `simple_list_item_1`. Die Layouts findet man unter `<path-to-sdk>/platforms/<android-platform>/data/res/layout`.



## Screenshot II



## Allgemeines

- ▶ GridView ist eine von ListView abgeleitete ViewGroup
- ▶ Stellt ein zweidimensionales, scrollbares Gitter bereit

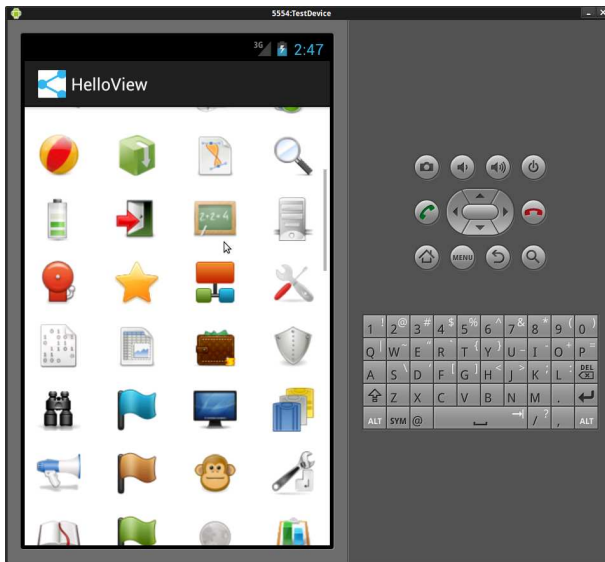
Attribut	Beschreibung
android:columnWidth	Breite einer Spalte [dimension]
android:gravity	Positionierung des Inhalts in einer Zelle [int]
android:numColumns	Anzahl der anzuzeigenden Spalten [int]
android:stretchMode	Legt fest, wie der Inhalt einer Zelle ausgedehnt werden soll, um die Zelle komplett auszufüllen [int]
android:horizontalSpacing	Horizontaler Abstand zwischen den Zellen [dimension]
android:verticalSpacing	Vertikaler Abstand zwischen den Zellen [dimension]

# Deklaration im Layout

```
<GridView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/icongrid"
  android:layout_width="match_parent"
5  android:layout_height="match_parent"
  android:columnWidth="70dp"
  android:numColumns="auto_fit"
  android:verticalSpacing="10dp"
  android:horizontalSpacing="10dp"
10 android:stretchMode="columnWidth"
  android:gravity="center" />
```

Listing : GridView

# Screenshot



## ExpandableListView

- ▶ Scrollbare Liste von Einträgen (wie ListView)
- ▶ Einzelne Einträge können aufgeklappt werden

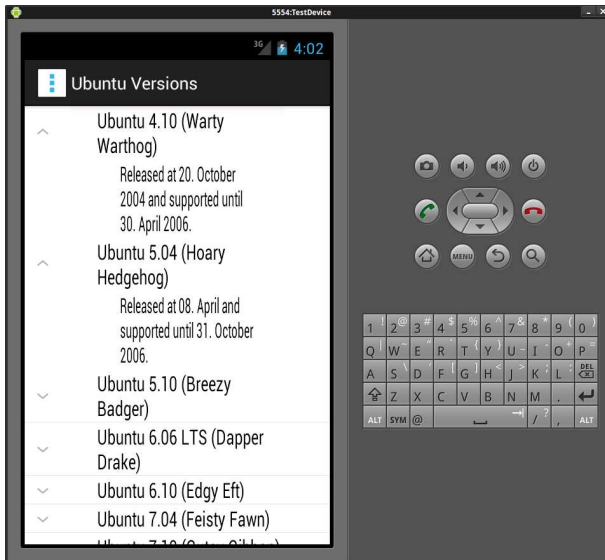
Attribut	Beschreibung
android:childDivider	Drawable oder eine Farbe zur Trennung der Kinder [resource]
android:childIndicator	Bild das neben dem Kindelement angezeigt wird [drawable]
android:childIndicatorLeft	Linke Begrenzung des Bildes für das Kindelement [int]
android:childIndicatorRight	Rechte Begrenzung des Bildes für das Kindelement [int]
android:groupIndicator	Bild das neben dem Gruppenelement angezeigt wird [drawable]
android:indicatorLeft	Linke Begrenzung des Bildes für das Gruppenelement [int]
android:indicatorRight	Linke Begrenzung des Bildes für das Gruppenelement [int]

## Deklaration im Layout

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
5   <ExpandableListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <TextView
10    android:id="@id/android:empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="No Data" />
</LinearLayout>
```

Listing : ExpandableListView

# Screenshot



# Adapter



## Allgemeines

- ▶ Von AdapterView abgeleitete Views müssen einen Adapter verwenden
- ▶ Adapter agiert als Brücke zwischen dem View und den anzuzeigenden Daten
- ▶ Standard-Adapter, wie ArrayAdapter, CursorAdapter oder SimpleCursorAdapter
- ▶ Adapter-Interfaces, wie ExpandableListAdapter
- ▶ Zuweisung eines Adapters mit *setAdapter()*
- ▶ Änderung an den Daten können über *notifyDataSetChanged()* mitgeteilt werden
- ▶ Allgemeine Klasse BaseAdapter bietet gute Grundlage

Methode	Beschreibung
int getCount()	Anzahl der Elemente
Object getItem(int position)	Zugriff auf Elemente anhand der Position
long getItemId(int position)	Zugriff auf ID der Elemente anhand der Position
View getView(int position, View convertView, ViewGroup parent)	Bereitstellung des Layouts für ein Element anhand der Position

## Deklaration eines Layouts

```

<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
5  android:orientation="horizontal"
  android:padding="7dip">
  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
10  android:src="@drawable/ubuntu"
    android:padding="3dip" />
  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
15  android:orientation="vertical">
    <TextView
      android:id="@+id/versionName"
      android:gravity="left|center_vertical"
      android:layout_height="wrap_content"
20  android:layout_width="match_parent"
      android:padding="7dip"
      android:singleLine="true" />
    <TextView
      android:id="@+id/supportDate"
      android:gravity="left|center_vertical"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
25  android:padding="7dip"
      android:singleLine="true"
      android:textScaleX="0.7" />
  </LinearLayout>
</LinearLayout>

```

Listing : Das Layout eines Eintrags

# Implementierung

```

public class UbuntuAdapter extends BaseAdapter {
    private String[] ubuntuVersions;
    private String[] ubuntuSupport;
    private LayoutInflater inflater;
5   private Resources res;

    public UbuntuAdapter(Context c) {
        this.res = c.getResources();
        this.inflater = (LayoutInflater) c.getSystemService(LAYOUT_INFLATER_SERVICE);
10       ubuntuVersions = this.res.getStringArray(R.array.ubuntu_array);
        ubuntuSupport = this.res.getStringArray(R.array.ubuntu_support_array);
    }

    public int getCount() {
15       return this.ubuntuVersions.length;
    }

    public String getItem(int position) {
20       return this.ubuntuVersions[position];
    }

    public long getItemId(int position) {
        return position;
    }
25

    public View getView(int position, View convertView, ViewGroup parent) {
        View row = this.inflater.inflate(R.layout.row_layout, null);

        TextView versionName = (TextView) row.findViewById(R.id.versionName);
30       versionName.setText(this.ubuntuVersions[position]);
        TextView supportDate = (TextView) row.findViewById(R.id.supportDate);
        supportDate.setText(this.ubuntuSupport[position]);

        return row;
35     }
}

```

# Screenshot

