Android – Eine Einführung

Entwicklungsumgebung, Aufbau, Komponenten & Widgets

Andreas Wilhelm

26. Juli 2014

Contents

Eclipse & Android SDK

Contents

Eclipse & Android

- Gemeinsame Oberfläche für verschiedene Entwicklungstools
- Basis für Android-Entwicklung – Android Software Development Kit (SDK)
- http://developer.android.com/sdk/
- Einbindung in Eclipse über das ADT-Plugin
- http://developer.android.com/tools/sdk/eclipse-adt.html

Das Android SDK

- Stellt Bibliotheken zur Android-Entwicklung zur Verfügung
- Zusätzliche Entwicklertools zum Testen & Debuggen
- Standard Support Library für Abwärtskompatibilität
- Ausführliche Anwendungsbeispiele
- Installer für Windows & Linux 32-bit verfügbar
- Unter Linux 64-bit muss *ia32-libs* Paket installiert werden

Der Emulator

- Simulation der Applikation unter verschiedenen Android Versionen
- Testen der Applikation bei verschiedenen Auflösungen
- Konfiguration verschiedener Geräte-Typen (Motorrola, HTC, ...)
- Verwalten verschiedener Konfigurationen über den Android-Virtuell-Device-(AVD)-Manager
- Automatisches Kompilieren, Uploaden und Starten der Applikation auf dem Emulator aus Eclipse heraus

Aufbau einer Applikation

Contents

Eigenschaften

Enthält grundlegende Informationen über eine Applikation wie

- Name, Paketname und Version der Applikation
- Verwendete Komponenten (Activities, Services und BroadcastReceiver)
- Zugangsmöglichkeiten zu den einzelnen Komponenten (Broadcasts, ...)
- Zugriffsrechte der Applikation (Internet, Anrufe, ...)
- Benötigte Zugriffsrechte anderer Applikationen zur Verwendung von Komponenten dieser Applikation
- Minimal und Maximal unterstützte API-Version
- Definition in Form einer eXtended-Markup-Language(XML)-Datei

Bispiel

```
<manifest>
    <uses-permission />
    <uses-sdk />
    <compatible-screens />

    <application>
        <activity>
            <intent-filter>...</intent-filter>
            <meta-data />
        </activity>

        <service>
            <intent-filter>...</intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter>...</intent-filter>
            <meta-data />
        </receiver>
    </application>
</manifest>
```

Listing : Beispielstruktur einer Manifest-Datei

Überblick

- Trennung von Quellcode und Ressourcen (Bildern, Zeichenketten, Layouts, ...)
- Eröffnet Möglichkeit verschiedenste Geräte zu unterstützen
- Verwaltung der Ressourcen in Verzeichnisstruktur unter *res*
- Feste Konventionen zur Einsortierung von Ressourcen (Beispiel: *res/values/strings.xml* und *res/values-de/strings.xml*)
- Zugriff auf Ressourcen aus dem Quellcode über automatisch generierte Klasse *R*
- Referenzierung der Ressourcen über IDs (Beispiel: *R.string.btnOk*)

```
<resources>
    <string name="app_name">TaskIt</string>
    <string name="hello_world">Hello world!</string>
    ...
</resources>
```

Listing : String Ressourcen

Beispiele

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="white">#ffffff</color>
    <color name="black">#000000</color>
    ...
</resources>
```

Listing : Farb Ressourcen

```java
public void onCreate(Bundle savedInstanceState) {
    // Call the parent method ...
    super.onCreate(savedInstanceState);

    // ... and set the content view.
    this.setContentView(R.layout.main);

    // Access the text view ...
    TextView moin = (TextView) this.findViewById(R.id.helloTxt);

    // ... and change the label.
    moin.setText(R.string.hello_world);
}
```

Listing : Zugriff auf Ressourcen

Überblick

- Globale Schnittstelle, die Zugriff auf wichtige Informationen der Applikation bzw. ihrer Umgebung ermöglicht
- Klasse *Context* ist abstrakt
- Zugriff auf Ressourcen
- Auslösen von Systemaufrufen
- Verbindung zwischen Komponenten und System
- Wichtige Methoden *getSharedPreferences()*, *getResources()*, *startActivity()*, *startService()* und *sendBroadcast()*

Implementierung

- *ContextWrapper* abgeleitet von *Context*
- *ContextThemeWrapper* abgeleitet von *ContextWrapper*
- *Activity* abgeleitet von *ContextThemeWrapper*

Komponenten in Android

Contents

Allgemeines

- Activities bilden die Präsentationsschicht einer Applikation
- Eine Applikation kann aus mehreren Activities (Dialoge oder Seiten) bestehen
- Implementierung einer Activity durch Ableitung einer Kindklasse von *Activity*
- Wichtig: Überschreiben der Methode *onCreate(Bundle)*
- Zuweisung der grafischen Oberfläche mittels der Methode *setContentView(View)*

```
public class TodoActivity extends Activity {
   protected void onCreate(Bundle savedInstanceState) {
      // Call the parent method ...
      super.onCreate(savedInstanceState);

      // ... and set the content view.
      this.setContentView(R.layout.main);
   }

   ...
}
```

Listing : Basisimplementation einer Activity

Lebenszyklus

Neben *onCreate()* gibt es weitere Callback-Methoden, die auf Zustandsänderungen der Activity reagieren. Man spricht vom *Activity Lifecycle*.
Aus der Stack basierten Struktur des Systems ergeben sich folgende Zustände für eine Activity:

Running Aktuell im Vordergrund befindliche Activity

Paused Sichtbare, aber nicht mehr im Fokus befindliche Activity – Activity bleibt aktiv und der WindowManager hat weiterhin Zugriff – Activity kann trotzdem bei Ressourcenmangel vom System beendet werden

Stopped Nicht sichtbare Activities – alle Einstellungen und Zustände gespeichert – werden häufig vom System beendet

Finished Vom System beendete Activity – muss neu gestartet werden und die Einstellungen der letzten Sitzung neu geladen werden, falls sie erneut den Fokus bekommt

Ablaufdiagramm

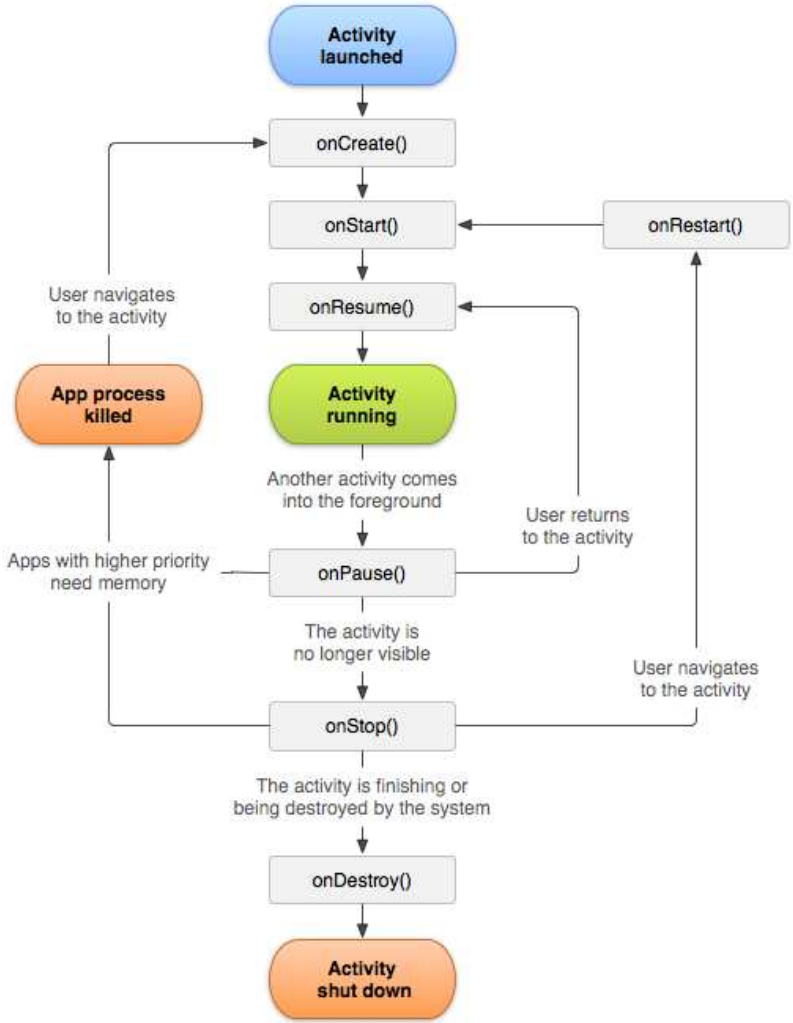


Abbildung: Der Activity Lifecycle (*Quelle: http://developer.android.com*)

Allgemeines

- Teil einer Activity – Eine Activity kann meherere Fragments enthalten
- Eingeführt in Android Version 3.0 (API-Version 11)
- Flexiblere grafische Oberflächen für größere Displays
- Einfacheres Austauschen und Wiederverwerten von Teilen der Oberfläche
- Verhalten ähnlich einer Activity – Lebenszyklen eng verbunden
  $\rightarrow$ Sollte Activity beendet werden, werden alle beinhalteten Fragments beendet

Zustände eines Fragments ähneln denen einer Activity:

Laufend Das Fragment ist sichtbar innerhalb der aktiven Activity.

Pausiert Die beinhaltende Activity ist zwar zumindest teilweise sichtbar, läuft aber im Hintergrund.

Gestoppt Entweder die beinhaltende Activity wurde beendet oder das Fragment wurde entfernt und auf dem ”Zurück”-Stapel (*Backstack*) der Activity abgelegt. Das Fragment ist jedoch, auch wenn nicht sichtbar, immernoch aktiv.

Lebenszyklus

Backstack

Android unterscheidet zwischen System- und Activity-Backstack. Activities werden automatisch auf den System-Backstack, Fragments nur nach expliziter Aufforderung auf Activity-Backstack abgelegt.

Neben von Activities bekannten Methoden, wie *onCreate()*, *onPause()* und *onResume()* implementieren Fragments weitere Methoden zur Synchronisation:

onAttach()	Fragment wird mit Activity assoziiert
onCreateView()	Layout des Fragments soll erstellt bzw. geladen werden
onActivityCreated()	Aufruf der Methode *onCreate()* der Activity wurde beendet
onDestroyView()	Ansicht wird aufgelöst
onDetach()	Assoziation zwischen Fragment und Activity wurde aufgehoben

Ablaufdiagramm

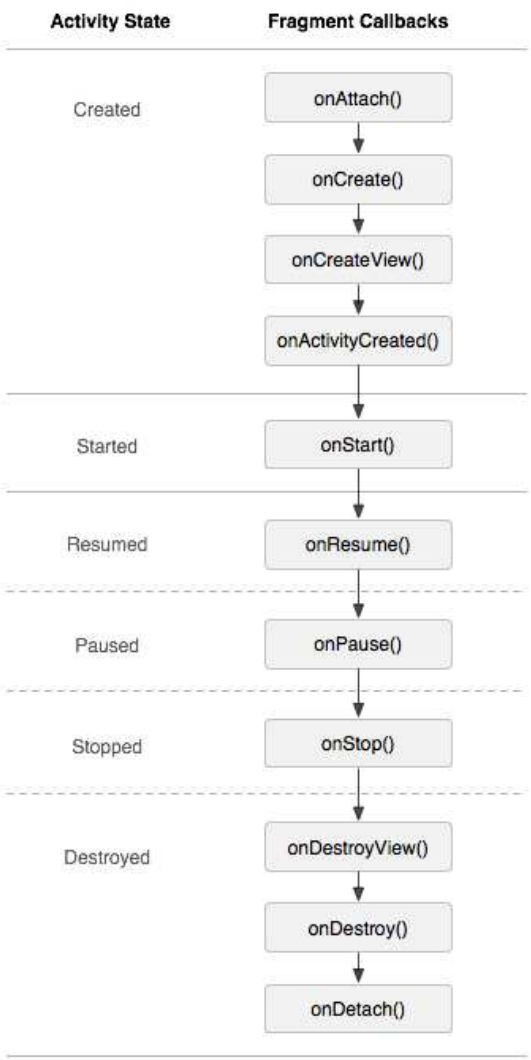


Abbildung: Der Lebenszyklus eines Fragments (*Quelle: http://developer.android.com*)

Implementierung

- Ableiten einer eigenen Klasse von *Fragment*, *ListFragment*, *PreferenceFragment* oder *WebViewFragment*
- Implementiert werden sollten *onCreate()*, *onCreateView()* und *onPause()*
- Wichtig: Änderungen sollten in *onPause()* gespeichert werden, denn es ist nicht sicher ob Benutzer zurückkehrt
- Einbettung des Fragments in Layout der Activity über *<fragment>*
- Alternativ Integration über Vater-ViewGroup des Fragments im Quellcode

IDs und Tags

Fragments sollten bei ihrer Integration mit *<fragment>* eine ID zugewiesen bekommen. Dies kann über die XML-Attribute *android:id* oder *android:tag* geschehen, aber auch automatisch durch Zuweisung zu einer ViewGroup, wobei die ID der ViewGroup verwendet wird.

Umrüstung

Die großen Ähnlichkeiten im Aufbau von Fragments und Activities, macht es zumeist sehr einfach eine Applikation auf Fragments umzurüsten. Oftmals reicht es den Quellcode aus einer Methode der Activity in die entsprechende Methode des Fragments zu verschieben.

Ein Fragment

```
public class OverviewFragment extends ListFragment {
   private ArrayAdapter<CharSequence> adapter;
   private Context context;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      // Setup the fragment ...
      super.onCreate(savedInstanceState);
      this.context = this.getActivity().getApplicationContext();

      // ... and assign the setup the release list adapter.
      this.adapter = ArrayAdapter.createFromResource(this.context, R.array.ubuntu_array,
            android.R.layout.simple_list_item_1);
      this.setListAdapter(this.adapter);
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
         Bundle savedInstanceState) {
      return inflater.inflate(R.layout.fragment_release_list, container, false);
   }

   @Override
   public void onPause() {
      // Nothing to save here.
      super.onPause();
   }

   @Override
   public void onListItemClick(ListView l, View v, int position, long id) {
      Toast.makeText(this.context, this.adapter.getItem(position), Toast.LENGTH_LONG).show();
   }
}
```

Listing : Das erste Fragment

Ein Fragment

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <fragment
        android:id="@+id/releaseList"
        android:name="net.avedo.ubuntu.releases.OverviewFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />
    <FrameLayout
        android:id="@+id/releaseInfo"
        android:layout_weight="2"
        android:layout_width="0px"
        android:layout_height="match_parent" />
</LinearLayout>
```

Listing : Einbinden des Fragments

Transactions

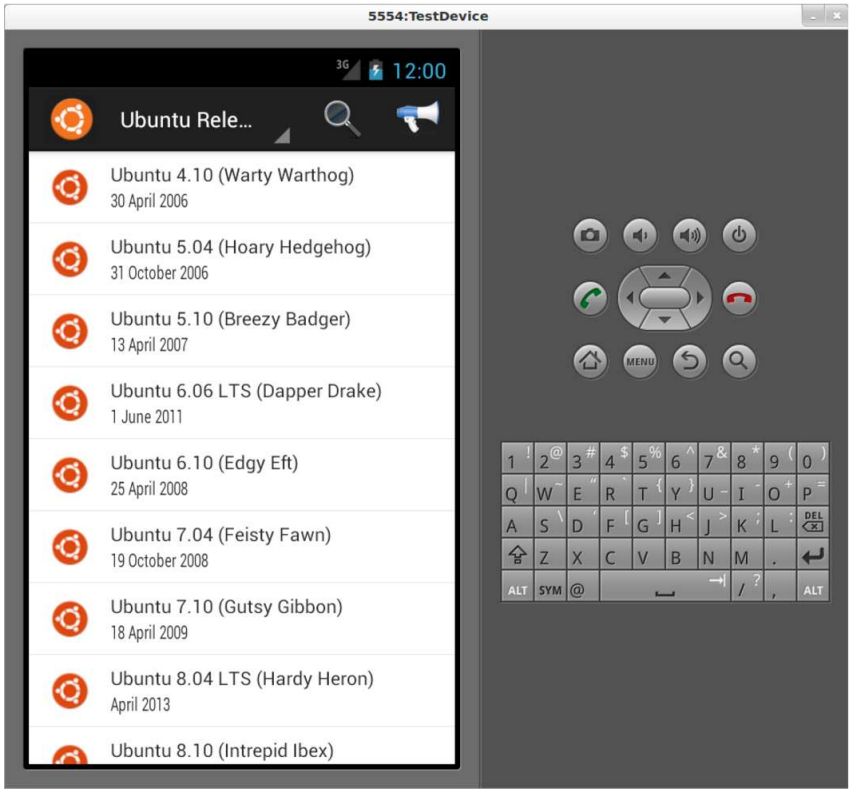
- Hinzufügen, Ersetzen und Löschen von Fragments in Activity
- Verarbeitung von *FragmentTransaction* mit *FragmentManager*
- Activity-Backstack verwaltet FragmentTransactions
- Einbinden des Fragments in Layout der Activity über *<fragment>*
- Alternativ Integration über Vater-ViewGroup des Fragments
- Mehrere Fragments zu Navigationszwecken
- Problem: Unterscheidung für große und kleine Displays nötig

Transactions im Einsatz

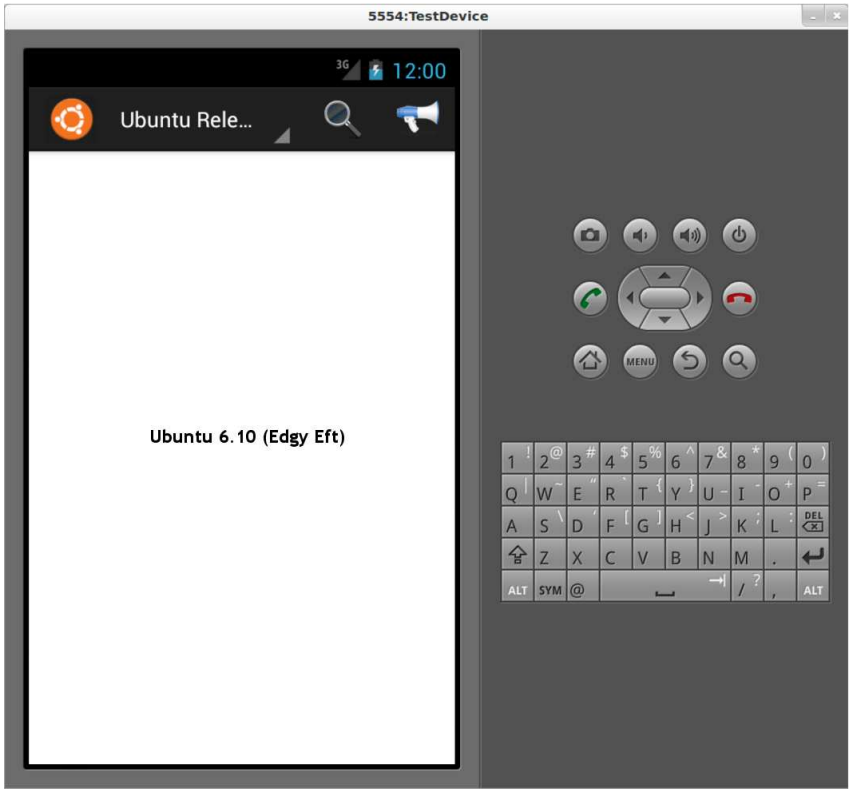
```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Fetch the FragmentManager object, ...
    FragmentManager fragmentManager = getFragmentManager();

    // ... instance the info fragment ...
    InfoFragment infoFrag = new InfoFragment();

    // ... and add the info data.
     Bundle args = new Bundle();
     args.putString(InfoFragment.INFO_DATA, this.adapter.getItem(position).toString());
     infoFrag.setArguments(args);

     // Finally start the transaction, ...
    FragmentTransaction fragmentTransaction = fragmentManager
            .beginTransaction();

    // ... replace the fragment within the releaseInfo FrameLayout, ...
    fragmentTransaction.replace(R.id.releaseInfo, infoFrag);

    // ... add this transaction to the backstack ...
    fragmentTransaction.addToBackStack(null);

    // ... and commit the transaction.
    fragmentTransaction.commit();
}
```

Listing : Eine FragmentTransaction

Activity-Lösung



(a) Erste Activity mit Liste



(b) Zweite Activity mit Zusatzinformationen

Abbildung: Navigation mit zwei Activities

Fragment-Lösung

Abbildung: Navigation mit zwei Fragments

Fragments ohne Layout

- Fragments ohne Layout als Erweiterungen von Activities
- Implementierung von Hintergrundarbeiten
- Implementierung von oft verwendeten Menüs
- Zuweisung von Fragments ohne Layout mit *add()*
- Laden dieser Fragments mit *findFragmentByTag()*
- *onCreateView()* wird nicht aufgerufen

Allgemeines

- Teile von Benutzeroberflächen, wie beispielsweise Buttons, Auswahl- oder Texteingabefelder
- Definition in Form einer XML-Datei
- Aussehen und Verhalten kann über XML-Attribute verändert werden
- *ViewGroups* (auch LayoutManager) dienen zur Strukturierung und Anordung von Views
- Eine beliebige Verschachtelung von ViewGroups ist möglich

Beispiel

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent" />
    <TextView
        android:id="@+id/subject"
        android:gravity="left|center_vertical"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textColor="@color/darkGrey"
        android:singleLine="true"
        android:textScaleX="1.3" />
</LinearLayout>
```

Listing : Ein einfaches Layout

Zugriff im Quellcode

- Zugriff auf Views und ViewGroups aus dem Quellcode heraus über *findViewById()*
- Laden eines Layouts ohne Verwendung von *setContentView()* mit *LayoutInflater*

```
// Fetch the layout inflater ...
this.inflater = (LayoutInflater) c.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

// ... and inflate the layout.
View main = this.inflater.inflate(R.layout.row_layout, null);

// Load the ImageView and assign the icon.
ImageView icon = (ImageView) main.findViewById(R.id.icon);
icon.setImageResource(R.drawable.my_icon);

// Load the TextView and assign the content.
TextView subject = (TextView) main.findViewById(R.id.subject);
subject.setText("Hello Bob");
```

Listing : Zugriff auf das Layout

Überblick

- Nachrichten zur Kommunikation innerhalb des Systems
- Aufruf von Funktionalitäten des Systems
- Übertragung von Daten zwischen verschiedenen Komponenten
- Methoden *startActivity()* und *startService()*, sowie *sendBroadcast()* nutzen Intents
- Unterscheidung zwischen expliziten und impliziten Intents

Explizite Intents

Der Empfänger der Nachricht wird explizit angegeben.

```
Intent i = new Intent(this, TodoActivity.class);
i.putExtra(TodoActivity.TODO_ID, 13);
startActivity(i);
```

Listing : Starten einer Activity

Die übertragenden Daten können in der *onCreate()* Methode extrahiert und verarbeitet werden.

```
protected void onCreate(Bundle savedInstanceState) {
   // Fetch the intent bundle, ...
   Bundle bundle = this.getIntent().getExtras();

   // ... initialize the todo id ...
   long todoId = Todo.PLACEHOLDER_ID;

   // ... and try to fetch it.
   if(bundle != null) {
      todoId = bundle.getLong(TodoActivity.TODO_ID, Todo.PLACEHOLDER_ID);
   }
}
```

Listing : Extrahieren der Daten

Implizite Intents

Anders als bei expliziten Intents, wird bei der Verwendung impliziter Intents nicht spezifiziert welche Komponente verwendet werden soll.

```
Intent smsIntent = new Intent(Intent.ACTION_VIEW);
smsIntent.setType("vnd.android-dir/mms-sms");
smsIntent.putExtra("sms_body", "Hello there! This is one way to send an sms.");
startActivity(Intent.createChooser(smsIntent, "Send an SMS");
```

Listing : Auswahl der Komponente

- System kann die richtige Komponente für die gegebene Aktion auswählen
- Man kann dem Benutzer die Möglichkeit geben eine andere Applikation zu nutzen
- Es kann auf andere Applikationen zurückgegriffen werden ohne die benötigte Funktionalität selbst zu implementieren

Ergebnisse von Intents

```
public class TodoList extends ListActivity {
   ...

   @Override
   protected void onListItemClick(ListView l, View v, int position, long id) {
      Intent i = new Intent(this, TodoActivity.class);
      i.putExtra(TodoActivity.TODO_ID, todo.getId());
      i.putExtra(TodoActivity.TODO_ACTION, TodoActivity.TODO_VIEW);
      startActivityForResult(i, TODO_VIEW_REQUEST);
   }
}
```

Listing : Starten der Activity

```
public class TodoActivity extends Activity {
   ...

   @Override
   public void finish() {
      Intent result = new Intent();
      result.putExtra(TodoActivity.TODO_ID, todo.getId());
      setResult(RESULT_OK, result);
      super.finish();
   }
}
```

Listing : Ergebnisse senden

Ergebnisse von Intents

```
public class TodoList extends ListActivity {
   ...

   @Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
      if (resultCode == RESULT_OK && requestCode == TODO_VIEW_REQUEST) {
         // Fetch the selected result.
         long id = data.getLong(TodoActivity.TODO_ID, Todo.PLACEHOLDER_ID);
      }
   }
}
```

Listing : Ergebnisse empfangen

IntentFilter

- Legen fest auf welche impliziten Intents eine Komponente anspricht
- Explizite Intents sind nicht von Filtern betroffen
- Ermöglicht Reaktion auf System-Events (ankommende SMS, Systemstart, ...)

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

    <application>
        <activity android:name="math.elearning.TodoList">
            <intent-filter>
                <action android:name="android.intent.action.SEARCH" />
            </intent-filter>
        </activity>

        <receiver android:name="math.elearning.CallLogObserver" android:enabled="true">
            <intent-filter>
                <action android:name="android.intent.action.PHONE_STATE" />
            </intent-filter>
        </receiver>

        ...
    </application>
</manifest>
```

Listing : Deklaration im Manifest

Allgemeines

- PendingIntents kapseln ein Intent und die zur Verwendung benötigten Rechte
- Delegation von Rechten an andere Applikationen bzw. das System
- Starten von Broadcasts und Activities
- *PendingIntent.getBroadcast()* bzw. *PendingIntent.getActivity()*
- Beispiel: NotificationManager

Beispiel

```
// Initialize the notification builder and assign the
// important notification data.
Notification.Builder releaseBuilder = new Notification.Builder(this)
    .setSmallIcon(R.drawable.ubuntu)
    .setContentTitle("New Ubuntu Release")
    .setContentText("Ubuntu 12.10 – Quantal Quetzal released!");

// Create an explicit intent to start the Ubuntu Release List activity.
Intent ubuntuIntent = new Intent(this, HelloList.class);

// Setup the pending intent to start the Ubuntu Release List activity.
PendingIntent pendingIntent = PendingIntent.getActivity(
      this, 0, ubuntuIntent, PendingIntent.FLAG_UPDATE_CURRENT);
releaseBuilder.setContentIntent(pendingIntent);

// Get the notification manager and show the notification.
NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(RELEASE_NOTIFICATION, releaseBuilder.build());
```

Listing : Benachrichtigung

Allgemeines

- Im Hintergrund arbeitende Komponenten für länger andauernde Operationen
- Bietet keine eigene Oberfläche
- Läuft im selben Thread wie die Applikation
- Können von einer anderen Komponente, Applikation oder durch ein Event (BroadcastReceiver) gestartet werden
- Kann an eine andere Komponente gebunden werden um mit ihr zu interagieren

Ein Service kann zwei Zustände annehmen:

Gestartet (Aufruf mit *startService()*)

- Beendet sich am Schluss selbst ohne ein Ergebnis zu liefern
- Kann praktisch unendlich laufen, auch wenn die startende Komponente beendet wurde

Gebunden (Aufruf mit *bindService()*)

- Service wurde von anderen Komponente mit *bindService()* an sich gebunden
- Interaktion (Nachrichten, Ergebnisse und Anfragen) mit startender Komponente möglich
- Bindung an beliebig viele Komponenten möglich
- Falls alle verbundenen Komponenten beendet oder getrennt werden, wird der Service beendet

Beispiel

```
public class TheService extends Service {
   MediaPlayer player;

   @Override
   public IBinder onBind(Intent intent) {
      return null; // No binding needed.
   }

   @Override
   public void onCreate() {
      Toast.makeText(this, "The Service started", Toast.LENGTH_LONG).show();

      player = MediaPlayer.create(this, R.raw.braincandy);
      player.setLooping(false); // Set looping
   }

   @Override
   public void onDestroy() {
      Toast.makeText(this, "The Service stopped", Toast.LENGTH_LONG).show();
      player.stop();
   }

   @Override
   public void onStart(Intent intent, int startid) {
      Toast.makeText(this, "The Service started", Toast.LENGTH_LONG).show();
      player.start();
   }
}
```

Listing : Ein einfacher Service

ContentProvider

- Verwalten den Zugriff auf Daten und stellen diese global zur Verfügung
- Kapseln die Daten und bieten Mechanismen zum Datenschutz
- *ContentResolver* ermöglichen die Kommunikation mit einem *ContentProvider* als Client
- ContentProvider werden beötigt, wenn Daten zwischen Applikationen kopiert werden sollen oder angepasste Suchergebnisse benötigt werden
- Es existieren Android eigene ContentProvider für Audio, Video, Bilder und Kontakte

BroadcastReceiver

- Empfänger von Intents die mit *sendBroadcast()* versendet wurden
- Kommunikation zwischen verschiedenen Komponenten
- Reagieren auf System-Events

```
<receiver android:name="canty.corby.SmsObserver" android:enabled="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

Listing : Deklaration im Manifest

```
public class SmsObserver extends BroadcastReceiver {
    private Handler handler = new Handler();

    @Override
    public void onReceive(Context context, Intent intent) {
        // Defer the processing a little bit.
        handler.postDelayed(new SmsReader(context, intent), 500);
    }
}
```

Listing : Implementierung

Widgets

Contents

Widgets

- Miniaturansicht, die in andere Applikationen, wie den Desktop integriert werden können
- Widgets werden periodisch aktualisiert
- Besteht aus folgenden Komponenten:

  AppWidgetProviderInfo XML-Datei, die für das System wichtige Informationen enthält

  AppWidgetProvider Implementiert die grundlegenden Methoden um mit dem Widget zu interagieren.

  Widget View Definiert das zu Anfang sichtbare Layout (XML-Datei)

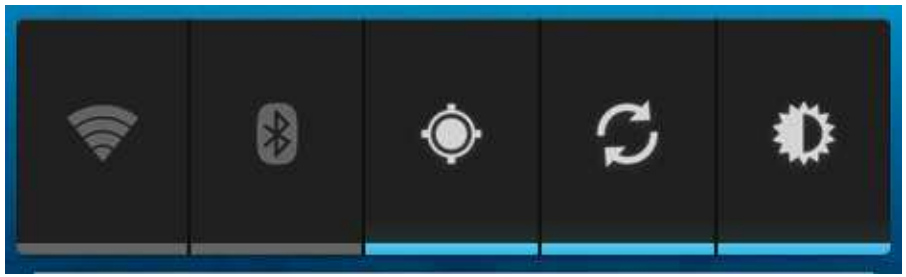


Abbildung: Das Einstellungs-Widget

Optional kann auch eine gesonderte Activity zur Konfiguration des Widgets hinterlegt werden.

Widget Layout

- Deklaration des Layouts in Form von XML
- Layouts werden in *res/layouts/* gespeichert
- Nicht alle Views und ViewGroups können verwendet werden (*NumberPicker*)

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/hello"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:background="@drawable/widget_shape"
        android:textColor="#ffffff" />
</FrameLayout>
```

Listing : Das Widget Layout

Die AppWidgetProviderInfo

- Definiert alle essentiellen Rahmenbedingungen eines Widgets (initiales Layout, Bemaßungen, Dauer bis zur Aktualisierung)
- Deklaration in Form einer XML-Datei

```
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="294dp"
    android:minHeight="72dp"
    android:updatePeriodMillis="300000"
    android:initialLayout="@layout/widget_layout"
    android:resizeMode="horizontal|vertical" />
```

Listing : Die Widget Info

Die Attribute des *appwidget-provider* Elements:

minWidth	Minimale Breite des Widgets in *dip*
minHeight	Minimale Höhe des Widgets in *dip*
updatePeriodMillis	Dauer bis zur nächsten Aktualisierung des Widgets in Millisekunden
initialLayout	Das initiale Layout
resizeMode	Definiert, wie die Bemaßungen durch den Benutzer angepasst werden können

Der AppWidgetProvider

- AppWidgetProvider ist von BroadcastReceiver abgeleitet
- Reagiert auf verschiedene Events und ruft entsprechende Methoden auf (FrontController Pattern)
- Broadcast werden ausgelöst, wenn ein Widget aktualisiert, gelöscht, aktiviert oder deaktiviert wird
- Folgenden Methoden verarbeiten die Events:

onUpdate()	Methode wird in regelmäßigen Abständen (*updatePeriodMillis*) aufgerufen – kümmert sich um die Aktualisierung des Widgets – sollte ein Konfigurationsdialog implementiert sein, muss dieser sich um den ersten Aufruf kümmern
onDelete()	Wird jedesmal aufgerufen wenn ein Widget dieses Typs gelöscht wird.
onEnabled()	Wird dann aufgerufen wenn ein Widget dieses Typs erzeugt wird (Kein zweites mal).
onDisabled()	Wird aufgerufen wenn das letzte Widget dieses Typs gelöscht wird (Bereinigung temporärer Daten)
onReceive()	Methode wird immer ausgeführt, wenn ein Broadcast empfangen wird (muss zumeist nicht implementiert werden)

Der AppWidgetProvider

```
public class HelloWidgetProvider extends AppWidgetProvider {
   public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
      final int numWidgets = appWidgetIds.length;
      final String[] helloStack = {
         "Hello", "Hi", "Shalom", "Ahlan", "Marhaba",
         "Terve", "Bula", "Aloha", "Salut", "Ciao",
         "Ave", "Hoi", "Morn", "Tschau", "Namaste"};

      // Loop over all instances of this widget.
      for (int i = 0; i < numWidgets; i++) {
         // Fetch the ListView, ...
         RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget_layout);

         // ... calculate stack position ...
         int rand = (int) (Math.random()*15);

         // ... and set the hello view.
         views.setTextViewText(R.id.hello, this.helloStack[rand]);

         // Perform an update on the current app widget.
         appWidgetManager.updateAppWidget(appWidgetIds[i], views);
      }
   }
}
```

Listing : Der AppWidgetProvider

Android 4

Neuerungen

- Weitgehend vereinheitlichtes GUI-Framework für Smartphones, Tablets und andere Endgeräte
- Neue Komponenten für die Erstellung von Benutzeroberflächen (NumberPicker, ...)
- Verbesserte Unterstützung größerer Displays
- Einfacher Zugriff auf Kontakte, Profildaten, Kalender-Einträge und soziale Netzwerke
- Verbesserte Multimedia API (Neue Codecs, erweitertes Streaming, ...)
- Verbesserung der Hardware-Anforderungen (Hardware beschleunigte 2D-Darstellung)
- Erweiterung von Widgets (AdapterViews werden nun Unterstützt)
- Neue VPN-API