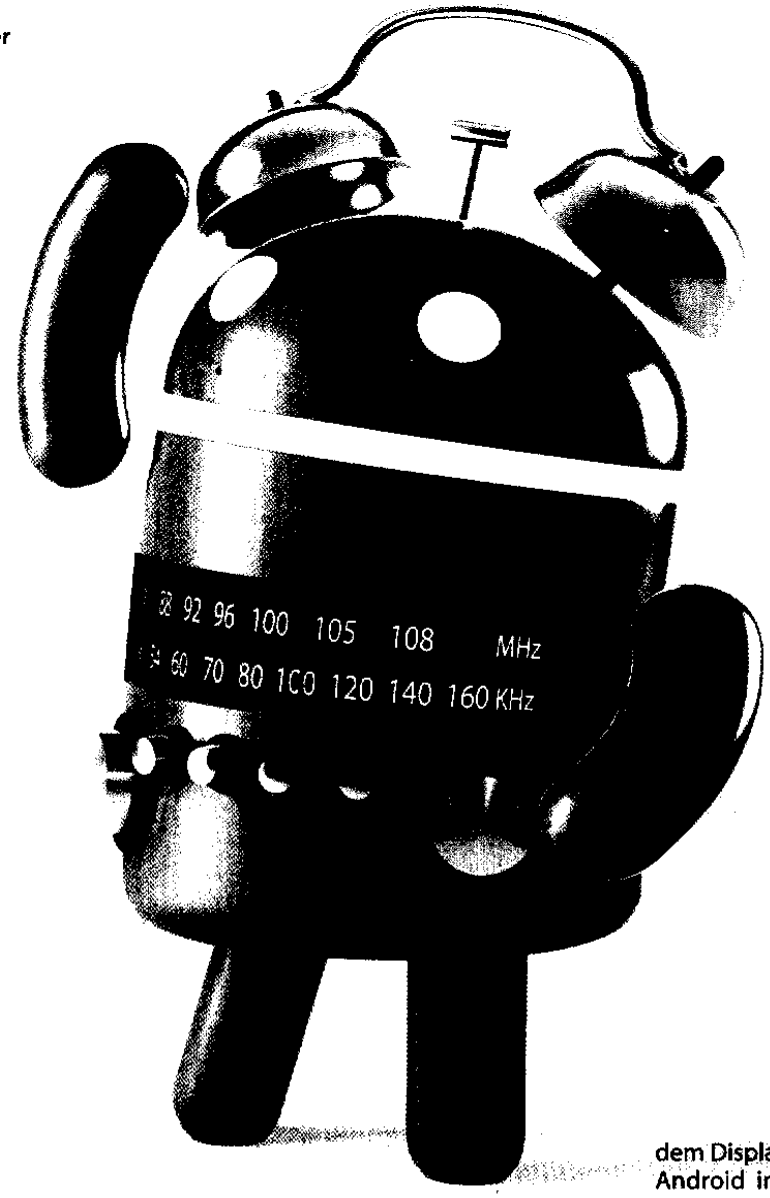


Andreas Linke

Wachmaschine

Radiowecker für Android programmieren

Smartphones und Tablets lassen sich prima als Radiowecker verwenden. Entsprechende Apps gibts zwar schon bei Google Play, aber entweder nerven sie mit Werbung oder sind Beta. Zum Glück ist eine eigene, nach Lust und Laune anpassbare Android-App leicht programmiert.

Ein Internetradio-Wecker, der besser als die eingebaute Uhren-App mit dem Lieblings-Radiosender weckt, lässt sich mit wenig Aufwand selbst entwickeln und an eigene Bedürfnisse anpassen. Nebenbei kann man daran einiges zum Umgang mit verschiedenen Android-APIs lernen, zum Beispiel für das Abspielen von Musik oder die Einplanung eines Alarms.

Die hier vorgestellte Wecker-App läuft auf Android ab 4.0 und erspart sich damit diverse Kompatibilitätskniffe. Mit der von Google bereitgestellten Support-Bibliothek können Sie sie jedoch auch auf älteren Android-Geräten zum Laufen bringen – die Kern-APIs MediaPlayer und AlarmManager sind in allen Versionen an Bord.

Wie ein echter Wecker besteht auch unsere App aus mehreren Teilen. Die Klasse MainActivity ist verantwortlich für die Anzeige der aktuellen Uhrzeit und die allgemeine Steuerung. Die InternetRadioActivity übernimmt das Einstellen von Sender und Lautstärke, während sich die AlarmActivity um die Programmierung der Weckzeiten je Wochentag kümmert. Die Hilfsklassen Alarm, Alarms und InternetRadio enthalten Funktionen zur Verwaltung und Aktivierung der Weckzeiten sowie zum Abspielen des Internetradio-Streams.

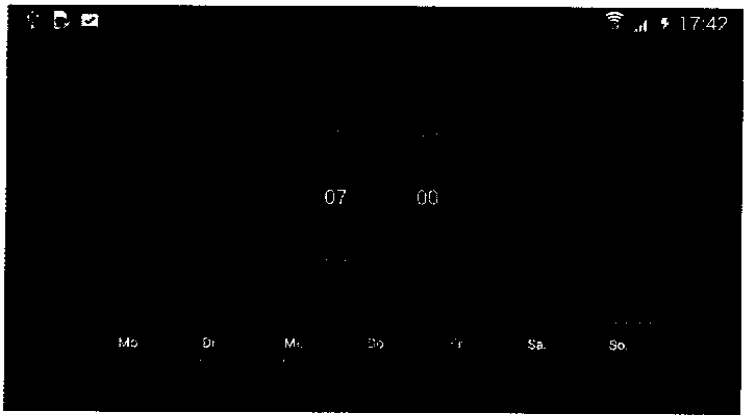
Zeiten

Das Wichtigste an einem Uhrenradio ist die Uhrzeit. Sie sollte in maximaler Größe auf dem Display angezeigt werden. Leider kennt Android im Unterschied beispielsweise zu iOS keine sich dynamisch an die Widget-Größe anpassenden Schriften. Abhilfe schafft eine kleine Hilfsroutine, die in einer Schleife ausgehend von einer maximalen Größe die Schrift so weit verkleinert, bis sie gerade nicht mehr abgeschnitten wird. Letzteres lässt sich durch einen Vergleich des Original-Textes mit dem Ergebnis der Funktion TextUtils.ellipsize() feststellen, die Text bei Übergröße abkürzt:

```
for (float textSize = height; textSize > 10;
    textSize *= 0.95) {
  textView.setTextSize(textSize); // in dips
  if (text.equals(TextUtils.ellipsize(text,
      textView.getPaint(), width, TextUtils.TruncateAt.END)))
    break;
}
```

Die App soll bis zu sieben Alarme verwalten können. Dazu speichert die Klasse Alarm die Uhrzeit zum jeweiligen Wochentag. Um den nächstmöglichen Alarmzeitpunkt auszurechnen, ist etwas Datumsarithmetik notwendig (siehe die Methode Alarm.nextAlarm() in Zusammenhang mit Alarm.calendar()). Von eigenen Berechnungen ist dringend abzuraten; selbst erfahrene Entwickler haben da schon gepatzt. So enthielt die erste Version von Android 4.2 einen peinlichen Fehler, der keine Eingaben von Geburtstagen und Feiertagen im Dezember erlaubte.

Wichtigste Regel bei der Arbeit mit Datum und Uhrzeit ist daher, immer die vom Be-



Die Weckzeit lässt sich für einen oder mehrere Wochentage gemeinsam einstellen.

Sie können den Link zum Stream entweder per Hand eingeben oder aus einer Liste deutscher Radiosender auswählen.

triebssystem bereitgestellten APIs zu verwenden. In Java übernimmt dies nicht die Date-, sondern die Calendar-Klasse. Sie enthält Konstanten für das Setzen einzelner Zeit- und Datumsbestandteile, kann beliebige Zeitspannen addieren und subtrahieren und mit Zeitzonen und Sommerzeitwechseln umgehen. Achtung: Stunden im 24-Stunden-Format sind mit Calendar.HOUR_OF_DAY zu setzen – verwendet man fälschlicherweise Calendar.HOUR, lassen sich nur Alarme am Vormittag erzeugen.

Für die Anzeige gibt es den SimpleDateFormatter, der sowohl verschiedene Darstellungsmöglichkeiten von Zeit und Datum gemäß den Benutzervorgaben kennt als auch einzelne Datumsteile wie den Wochentag in lokalisierter Form liefern kann.

Zum Speichern der Alarmzeiten bietet sich das JSON-Format an, denn es ist kompakt und plattformübergreifend austauschbar. Für die Serialisierung komplizierterer Java-Objekte wird häufig das Open-Source-Framework GSON verwendet; die einfachen Strukturen in dieser App lassen sich aber schon mit den in Android verfügbaren JSONObject-Klassen speichern. Den fertigen JSON-String nimmt ein SharedPreference-Objekt auf und speichert es im Dateisystem der App.

Eine elegante und übersichtliche Oberfläche für die Programmierung von Weckzeiten ist gar nicht so einfach, wie die häufig auf kommerziellen Weckern zu findenden Knöpfchenorgien beweisen. Wir haben uns für eine Kombination aus einer Liste und einem Fragment mit TimePicker und Wochentagsliste entschieden. Letztere wird als Gruppe von ToggleButtons dargestellt. Im Standard haben diese einen unschön weiten Abstand – ein negativer Randabstand (Margin) von –5dp lässt sie enger zusammenrücken. Alarme, die zur gleichen Uhrzeit beginnen (etwa für die Werktage), werden automatisch in einer Gruppe zusammengefasst, sodass sich die Weckzeit schnell ändern lässt.

Eine ordentlich lokalisierte App berücksichtigt übrigens nicht nur das vom Benutzer in den Android-Einstellungen unter „Datum & Uhrzeit" vorgegebene Format, sondern auch den richtigen Beginn der Woche (Montag in Europa, Sonntag in Amerika). Diesen liefert Calendar.getFirstDayOfWeek() (1 ist Sonntag, 2 ist Montag und so weiter).

Nachrichten

Einen Alarm, der die App zu einem bestimmten Zeitpunkt aufweckt, erzeugt man mit dem AlarmManager-Service. Er verlangt einen sogenannten PendingIntent. Das ist eine spezielle Klasse, die einen in der Zukunft aufzurufenden Intent verpackt. Der Intent muss auf eine von BroadcastReceiver abgeleitete Klasse zeigen. Außerdem benötigt man für das Setzen des Alarms die Zeit als Millisekunden seit dem 1. Januar 1970, 0 Uhr UTC. Die Funktion Calendar.getTimeInMillis() liefert den passenden Wert:

```
PendingIntent pi = PendingIntent.getBroadcast(context,
  0, new Intent(context, AlarmReceiver.class), 0);
AlarmManager am = (AlarmManager)context
  .getSystemService(Context.ALARM_SERVICE);
am.set(AlarmManager.RTC_WAKEUP,
  alarm.calendar().getTimeInMillis(), pi);
```

Empfänger der Aufweck-Nachricht ist eine von BroadcastReceiver abgeleitete Klasse, die im Android-Manifest eingetragen sein muss:

```
<receiver android:name=".AlarmReceiver" />
```

In ihrer Methode onReceive() reagiert sie auf die Benachrichtigung und startet die MainActivity.

Ist der Alarmzeitpunkt erreicht, wird das Gerät aufgeweckt und der registrierte Empfänger benachrichtigt. Ist das geschehen, kann das System theoretisch sofort wieder in den Schlafmodus gehen. Um das zu verhindern, muss die App eine Weck-Sperre (WakeLock) anfordern:

```
PowerManager powermgr = (PowerManager)
  context.getSystemService(Context.POWER_SERVICE);
wakeLock = powermgr.newWakeLock(
  PowerManager.FULL_WAKE_LOCK |
  PowerManager.ACQUIRE_CAUSES_WAKEUP,
  "RadioAlarmClock");
wakeLock.acquire(3600 * 1000); // in ms
```

Damit die App nicht unbegrenzt angeschaltet bleibt und so die Batterie leer saugt, bekommt das Wake-Lock einen Timeout von einer Stunde.

Weck-Sperren funktionieren nur mit der Erlaubnis

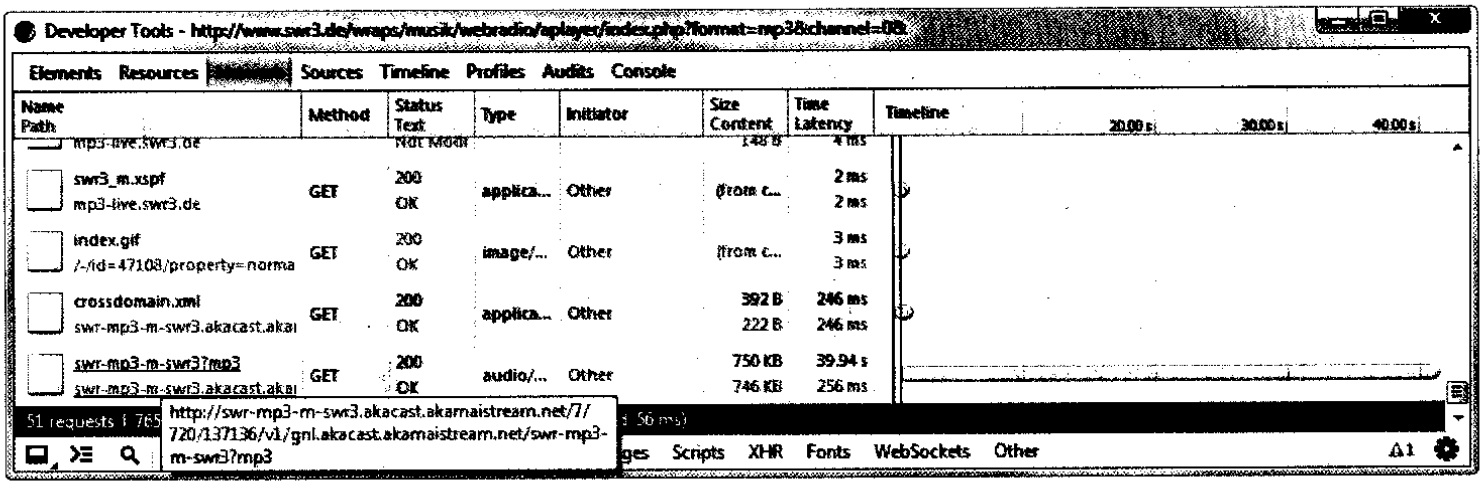
```
<uses-permission android:name=
  "android.permission.WAKE_LOCK" />
```

im Manifest.

Außerdem soll die App den Bildschirm einschalten und die Activity bei gesperrtem Bildschirm anzeigen. Das geschieht über das Setzen der entsprechenden Flags:

```
getWindow().addFlags(
  WindowManager.LayoutParams
    .FLAG_SHOW_WHEN_LOCKED |
  WindowManager.LayoutParams
```

Die Bedienoberfläche des Radioweckers ist bewusst minimalistisch gehalten.



Der Inspektor in Chrome hilft beim Aufspüren der URL des Lieblingsradio-Streams.

```
    .FLAG_TURN_SCREEN_ON |
  WindowManager.LayoutParams
    .FLAG_KEEP_SCREEN_ON);
```

Nach einem Reboot des Geräts werden alle Alarme gelöscht. Zum Glück gibt es für den Neustart des Geräts die Notification android.intent.action.BOOT_COMPLETED, auf die sich die App über einen Intent-Filter im Manifest registrieren kann. Sie empfängt dieses Signal und setzt im Handler den Alarm nach dem Booten erneut.

Ab Android 4.4 kann der Aufweckzeitpunkt übrigens ein wenig von der angegebenen Uhrzeit abweichen. Das Betriebssystem kann so Energie sparen, etwa indem es mehrere Apps zur gleichen Zeit aufweckt. Für den hier beschriebenen Verwendungszweck ist die Genauigkeit auf jeden Fall hoch genug. Bei unseren Versuchen betrug die maximale Abweichung nur einige 100 Millisekunden.

Radiologie

Googles Play Store ist voll von Internetradio-Apps. Android sieht prinzipiell den Aufruf anderer Apps über Intent-Filter vor. Leider halten sich die Hersteller oft bedeckt, wenn es um technische Details dazu geht. Eine Fernsteuerung Ihrer Lieblings-Internetradio-App lässt sich daher nicht ohne Weiteres allgemeingültig programmieren. Zum Glück ist es nicht schwer, einen Internetradio-Stream direkt in die App zu integrieren. Die Android-Klasse MediaPlayer bringt alle notwendigen Funktionen bereits mit. Das Abspielen ist mit wenigen Zeilen erledigt:

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(streamURL);
mp.setOnPreparedListener(new OnPreparedListener() {
  @Override
  public void onPrepared(MediaPlayer mp) {
    mp.start();
  }
});
mp.prepareAsync();
```

Die Fehlermeldung „Should have subtitle controller already set", die dabei auf der Konsole ausgegeben wird, ist ein Fehler in Android 4.4, den Sie getrost ignorieren können.

Da die prepare()-Phase, in der die Internet-Verbindung hergestellt wird und die ersten Daten gepuffert werden, einige Sekunden dauern kann, wird sie asynchron ausgeführt. Ein Callback erhält Bescheid, wenn der MediaPlayer zum Abspielen bereit ist. Erst dann darf die App start() aufrufen.

Die Klasse InternetRadioActivity der Beispiel-App übernimmt die Eingabe und Anzeige der Stream-URL des Senders sowie die Einstellung der Wecklautstärke. Da sie nur wenige Steuerelemente hat und nicht den ganzen Bildschirm benötigt, wird sie einfach als Popup angezeigt. Das lässt sich im Manifest mittels

```
android:theme="@android:style/Theme.Holo.Dialog"
```

festlegen.

Wie kommt nun der Lieblingssender in die App? Zunächst wird eine Stream-URL benötigt. Diese URL findet man zum Beispiel im Chrome-Browser, wenn man dort vor dem Aufruf einer Radioseite den Inspektor mit Strg+Umsch+I einschaltet und dann im Network-Tab auf den kontinuierlich laufenden Request klickt. Es gibt auch Übersichtsseiten im Web, die umfangreiche Listen mit Internetradio-Streams pflegen. Wir haben in die App einen Link auf eine solche Website (www.listenlive.eu/germany.html) eingebaut, die mehrere hundert deutschsprachige Internetradio-Streams in unterschiedlicher Qualität auflistet. Von dort lässt sich die Stream-URL des gewünschten Radiosenders als Playlist-Datei (.m3u oder .pls) herunterladen und mit der Radio-App öffnen. Nicht alle Links funktionieren unter Android, probieren Sie im Zweifel einfach verschiedene Bitraten aus.

Damit die Beispiel-App eine Playlist öffnen kann, registriert sich die InternetRadioActivity im Manifest mit einem Intent-Filter für alle Dateien des entsprechenden Formats:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name=
    "android.intent.category.BROWSABLE" />
  <category android:name=
    "android.intent.category.DEFAULT" />
  <data android:scheme="http" />
  <data android:scheme="https" />
  <data android:host="*" />
  <data android:pathPattern=".*\\.m3u" />
  <data android:pathPattern=".*\\.pls" />
</intent-filter>
```

Bedingungen mit verschiedenen Datentypen (beispielsweise android:scheme oder android:pathPattern) werden Und-verknüpft, mehrfach aufgeführte Bedingungen des gleichen Datentyps hingegen Oder-verknüpft. Der obige Filter wirkt also auf URLs mit http- oder https-Schema, aber nur für Dateien, die auf .m3u oder .pls enden.

Auch wenn es bei einem flüchtigen Blick so aussieht als ob, erlaubt Android für das pathPattern keine regulären Ausdrücke. Das Muster „.*\\.m3u" funktioniert zum Beispiel nicht, wenn der Dateiname einen weiteren Punkt enthält, etwa „tide-40.mp3.m3u". Die etwas umständliche Lösung besteht darin, zusätzliche Muster mit weiteren Punkten aufzunehmen, beispielsweise „.*\\..*\\.m3u".

Nun muss die App noch die Playlist herunterladen und auslesen. Das geschieht mit Hilfe der Klasse HttpURLConnection, und zwar im Hintergrund, damit die Bedienoberfläche während des Downloads interaktiv bleibt:

```
AsyncTask<Uri, Void, Void> asyncTask = new
  AsyncTask<Uri, Void, Void>() {
    @Override
    protected Void doInBackground(Uri... uri) {
      final String streamURL = openURI(uri[0]);
      // ...
    }
  };
asyncTask.execute(uri);
```

Die in der heruntergeladenen Datei enthaltene Stream-URL lässt sich mit einem einfachen regulären Ausdruck herauslösen:

```
Matcher matcher = Patterns.WEB_URL.matcher(data);
if (matcher.find()) {
  streamURL = matcher.group();
  // ...
```

Mit der Activity-Hilfsfunktion runOnUiThread(new Runnable()) wird dann aus dem Hintergrundprozess heraus die Oberfläche aktualisiert:

```
runOnUiThread(new Runnable() {
  @Override
  public void run() {
    editStreamURL.setText(streamURL);
    // ...
  }
});
```

Krachmacher

Für die Lautstärke der abgespielten Musik ist der AudioManager-Service zuständig. Er erlaubt das Auslesen und Setzen der Lautstärke getrennt für verschiedene Audio-Typen wie Musik, Wecker oder Systemtöne. Zum Beispiel stellt

```
AudioManager audioManager = (AudioManager)
  getSystemService(Context.AUDIO_SERVICE);
audioManager.setStreamVolume(
  AudioManager.STREAM_MUSIC, volume, 0);
```

die Lautstärke für das Abspielen von Musik ein. Der maximal mögliche Wert lässt sich über audioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC) abfragen.

Da sich die Lautstärkeänderung auf das gesamte Gerät bezieht, sollte eine ordentliche App den ursprünglich vom Benutzer vorgegebenen Wert in onResume() speichern und in onPause() wiederherstellen.

Etwas verwirrend ist vielleicht, dass sich die Lautstärke auch mit MediaPlayer.setVolume(float, float) einstellen lässt, und zwar getrennt für beide Stereokanäle. Normalerweise setzt man beide auf den Maximalwert 1. Für die tatsächliche Lautstärke werden die mit MediaPlayer.setVolume() und AudioManager.setStreamVolume() gesetzten Werte multipliziert.

Damit das Radio nicht schlagartig mit voller Lautstärke losplärrt, erhöht die App über einen Timer die Lautstärke langsam bis zum eingestellten Maximalwert:

```
final Timer t = new Timer("RadioVolume");
t.schedule(new TimerTask() {
  @Override
  public void run() {
    int volume = audioManager
      .getStreamVolume(AudioManager.STREAM_MUSIC)
      + deltaVolume;
    audioManager.setStreamVolume(
      AudioManager.STREAM_MUSIC, volume, 0);
    if (volume >= targetVolume)
      t.cancel();
  }
}, 0, 1000);
```

Verschlafen, weil keine Internet-Verbindung hergestellt werden konnte, zählt wohl kaum als Entschuldigung. Ein vertrauenswürdiger Wecker hat für diesen Fall einen Plan B: Die Beispiel-App prüft, ob der Internet-Stream tatsächlich abgespielt werden konnte (also das Callback onPrepared() aufgerufen wurde). Passiert dies nicht innerhalb von 20 Sekunden (siehe MainActitity.onAlarmUp()), wird auf den Standard-Weckton zurückgegriffen:

```
RingTone ringtone =
RingtoneManager.getRingtone(context,
  RingtoneManager.getDefaultUri(
    RingtoneManager.TYPE_ALARM));
ringtone.setStreamType(AudioManager.STREAM_ALARM);
ringtone.play();
```

Und weiter ...

Fertig ist die App, die ein Android-Gerät in einen persönlichen Radiowecker verwandelt. Mit dem Gehäuse aus [1] kann daraus sogar ein richtiger Hingucker auf dem Nachttisch werden.

Die vorgestellte App lässt sich leicht erweitern oder verbessern: Wer ein Gerät mit Android 4.2 oder höher besitzt, kann die Uhrzeit auch bei schlafendem Gerät als DreamService anzeigen. Einzelheiten dazu finden Sie bei den Fußball spielenden Androiden aus [2]. Die schlichte Digital-Anzeige des Beispiels ist vielleicht nicht jedermanns Geschmack – eine analoge Uhr lässt sich mit relativ wenig Aufwand auf einem Canvas zeichnen. Die Fraktale-App aus [3] leistet hier technische Hilfestellung. (ola)

Literatur

[1] Jan Schüßler, Smarte Box, Internet Radio im Selbstbau, c't 18/13, S. 130

[2] Andreas Linke, Tagträumer, Animierte Bildschirmschoner für Android programmieren, c't 7/13, S. 182

[3] Andreas Linke, Appfelmännchen, Grafiken, Threads und C-Programme auf der Android-Plattform, c't 5/11, S. 188

[4] Andreas Linke, Aufs Tablet gebracht, Programmieren für Android 3, c't 7/12, S. 190

[5] Android-API-Referenz, Build.VERSION_CODES: http://developer.android.com/reference/android/os/Build.VERSION_CODES.html

www.ct.de/1404182

Apps fit machen für verschiedene Android-Versionen

Aktuelle Android-SDKs bringen inzwischen ein ganzes Bündel von Hilfswerkzeugen mit, um der Vielfalt der Android-Versionen Herr zu werden.

Google hat eine Support-Bibliothek zusammengestellt, die moderne Android-Features wie Fragmente oder den ActionBar auf älteren Android-Versionen nachbildet. Sie wird standardmäßig beim Anlegen eines neuen Projektes hinzugefügt [4].

Ein weiteres mächtiges Feature ist der Android-Code-Checker (Android-Lint), der häufige Fehler im Quellcode wie auch in den Layout-XMLs aufspürt und unter anderem prüft, ob alle verwendeten APIs mit der im Manifest vorgegebenen minSdkVersion kompatibel sind. Ausnahmen erlaubt man explizit über die Annotation @TargetApi an der Funktion oder Klasse – natürlich nur, wenn man sicher ist, dass der Code auch wirklich erst ab der entsprechenden Android-Version ausgeführt wird. Mit der QuickFix-Funktion (Strg+1) von Eclipse lässt sich die Annotation mit wenigen Klicks einfügen.

Es empfiehlt sich, stets wirklich alle Warnungen des Android-Lint-Checkers zu beseitigen. Nur dann fällt bei der Weiterentwicklung sofort auf, wenn versehentlich ein in älteren Android-Versionen nicht vorhandenes API benutzt wird. Die einzelnen Lint-Warnungen lassen sich unter „Window/Preferences/Android/Lint Error Checking" ein- und ausschalten, die Voreinstellung passt aber in den meisten Fällen bereits ziemlich gut und sollte möglichst nicht verändert werden.

Schließlich gibt es mit minSdkVersion, targetSdkVersion und maxSdkVersion die Möglichkeit, die Installierbarkeit und das Verhalten der App unter verschiedenen Android-Versionen zu steuern. Während minSdkVersion selbsterklärend ist, und maxSdkVersion nur selten verwendet wird (zum Beispiel wenn man explizit verschiedene Versionen einer App baut und diese als Multiple APKs in den Google Play Store lädt), lohnt es sich, genauer auf die targetSdkVersion zu schauen. Diese hat nämlich Einfluss darauf, wie sich die App auf dem Gerät verhält. Ähnlich wie Web-Browser fällt das Android-Betriebssystem in einen Kompatibilitätsmodus zurück, wenn es eine ältere Target-SDK-Version in einer App findet. So ist es etwa ab Android 3.0 (Honeycomb) nicht mehr erlaubt, im UI-Thread auf das Netzwerk zuzugreifen – Apps, die mit älteren Target-SDK-Versionen entwickelt wurden und das nicht beachten, laufen dank Kompatibilitätsmodus trotzdem.

Was sich für eine App zu einem bestimmten TargetSDK ändert, steht im Detail unter [5]. Es lohnt sich, beim Aktualisieren der Target-Version auf das neueste Release die entsprechenden Änderungen sorgfältig durchzulesen und zu testen.

c't