Texte definieren

Um die Verweise auf Ressourcen aufzulösen, definieren wir die Texte in der Datei `/res/values/strings.xml`. Dort legen wir die folgenden Einträge an:

```
<resources>
  <string name="startseiteanzeigen_intro">
      Sie haben folgende Optionen zur Verfügung:
  </string>
  <string name="app_positionSenden">
      Position senden
  </string>
</resources>
```

Farben definieren

Ebenso legen wir in der Datei `/res/values/colors.xml` die referenzierten Farbwerte an:

```
<resources>
  <color name="hintergrund">#FFFFFF</color>
  <color name="textfarbe">#000000</color>
</resources>
```

Listing 5.11 setzt ebenfalls voraus, dass im Ordner `drawable` eine Datei mit Namen *amando_logo* liegt. Wir haben nun die Verbindung von Ressourcen, Layout und Views hergestellt und eine komplette Anzeigeseite entwickelt. Unsere Schaltflächen sind jedoch noch ohne Funktion. Dies ist unter anderem Thema des nächsten Abschnitts.

5.5 Schaltflächen und Menüs

Mit zunehmender Beliebtheit von berührungsempfindlichen Bildschirmen (engl. *touch screens*) im Mobilgerätebereich stellt sich für Entwickler die Frage, auf welche Weise sie die Navigation zwischen den Bildschirmseiten am anwenderfreundlichsten gestalten.

Qual der Wahl

Es gibt zwar häufig noch die vom Mobiltelefon bekannte Navigation über die Menütaste des Telefons. Man kann sich aber nicht darauf verlassen, dass sie immer vorhanden sind. Außerdem ist die Steuerung über Schaltflächen in Verbindung mit den neuartigen Bildschirmen oder Steuerungskomponenten (Cursor, Track-Ball etc.) in den meisten Fällen einfacher zu bedienen. Probleme treten hier erst auf, wenn zu viele Auswahlmöglichkeiten angeboten werden müssen. Eine Bildschirmseite mit vielen Schaltflächen wird schnell unübersichtlich. Für unser Projekt legen wir daher die folgende Implementierungsrichtlinie fest:

Wir ziehen die Verwendung von Schaltflächen zur Bildschirmnavigation der klassischen Menütasten-Navigation vor, wenn auf dem Bildschirm genug Platz dafür ist. Die Steuerung über Menütasten sollte nur für Standardfunktionen (Einstellungen ändern, Programm beenden etc.) verwendet werden.

5.5.1 Schaltflächen

Am Ende des Abschnitts Views haben wir die Definition von Schaltflächen schon kennengelernt. Hier ein Ausschnitt des `Startseiten`-Layouts zur Auffrischung:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

  <Button
    android:id="@+id/sf_starte_geokontakte"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/sf_geokontakteVerwalten" />

</LinearLayout>
```

Piktogramm := Symbolbildchen

Statt eines Textes können Schaltflächen ein Piktogramm (engl. *icon*) anzeigen. Dazu definiert man anstatt eines `Button` im Layout einen `ImageButton`, wie im folgenden Beispiel:

```
  <ImageButton
    android:id="@+id/sf_starte_geokontakte"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/notification_icon" />
```

Sagen Bilder mehr als Worte?

Der Ressourcenschlüssel `@drawable/notification_icon` verweist auf die Piktogramm-Datei `notification_icon.png` im Ordner `drawable-mdpi`. Falls aussagekräftige Piktogramme zur Verfügung stehen, sollte man diese gegenüber Textbeschriftungen vorziehen, nicht zuletzt wegen der Platzersparnis.

Piktogramme selbst machen

Falls Sie eigene Piktogramme entwerfen wollen, finden Sie Hilfestellungen in der Android-Referenz. Die Richtlinie zur Gestaltung von Piktogrammen (engl. *Icon Design Guidelines*) [22] beschreibt eine Vorgehensweise zum Erstellen verschiedener Piktogrammarten. Für Anwender von Photoshop und Illustrator steht außerdem ein Paket von Vorlagen als »Android Icon Template Pack« zum Herunterladen bereit.

Der `ImageButton` ist nett, jedoch leider ohne Funktion. Das wollen wir ändern: Bei Betätigung der Schaltfläche wollen wir im Quelltext eine Aktion ausführen. An dieser Stelle soll später die Bildschirmseite mit der Liste der Geokontakte aufgerufen werden.

Seit Android API 1.6 ist das mit sehr wenig Quelltext umsetzbar; das API bietet dafür das XML-Attribut `android:onClick` an, in dem die aufzurufende Methode der Activity angegeben wird.

Wir definieren zunächst eine Methode mit aussagekräftigem Namen in der `Startseite`-Activity:

```
public void onClickGeokontakteVerwalten(
    final View sfNormal) {

  // rufe GeoKontakte verwalten auf...
}
```

Anschließend ergänzen wir die View-Definition um das `onClick`-Attribut.

```
<ImageButton
  android:id="@+id/sf_starte_geokontakte"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:src="@drawable/notification_icon"
  android:onClick="onClickGeokontakteVerwalten" />
```

Bei Android-API-Versionen kleiner 1.6 ist ein wenig mehr Aufwand vonnöten.

5.5.2 Oberflächenereignisse

Es gibt nur einen Beobachter.

Einen Klick auf eine Schaltfläche schickt Android als Ereignis an den registrierten Beobachter (engl. *Listener, Observer*). Die in einem solchen Beobachter-Interface deklarierten Methoden bezeichnet man auch als *Callback-Methoden*. Die wichtigsten Ereignisbeobachter sind in Tabelle 5-22 aufgelistet.

Callbacks

Ereignisbeobachter	wird benachrichtigt, ...
View.OnClickListener	wenn die View angeklickt wird.
View.OnLongClickListener	wenn die View länger angeklickt wird.
View.OnKeyListener	bevor eine Tastatureingabe an die View geschickt wird.
View.OnTouchListener	wenn der berührungsempfindliche Bildschirm (Touch-Screen) einen Klick auf die View meldet.

Tab. 5-22 *Einige Ereignisbeobachter der Android-API*

Vor Android 1.6 Für eine Implementierung in einer Android API vor 1.6 definieren wir für unser Beispiel zunächst einen `View.OnClickListener`-Beobachter in der `Startseite` Activity.

```
private OnClickListener mGeokontakteVerwaltenListener =
    new OnClickListener() {
  public void onClick(View v) {
    onClickGeokontakteVerwalten(v);
  }
};
```

Nun fehlt noch die Registrierung an der Schaltfläche. Dazu fügen wir den folgenden Code in der `onCreate`-Methode der Activity ein:

```
final Button buttonGeokontakteVerwalten =
    (Button) findViewById(R.id.sf_starte_geokontakte);
buttonGeokontakteVerwalten
    .setOnClickListener(mGeokontakteVerwaltenListener);
```

Ereignisse dürfen verbraucht werden.

Ähnlich ist für andere Oberflächenereignisse vorzugehen, mit einem Unterschied: Die anderen Methoden in der Tabelle geben einen `boolean` zurück. Damit signalisiert die Implementierung, ob das Ereignis verbraucht (engl. *consumed*) wurde. Eine Rückgabe von `true` beendet die Ereignisbearbeitung nach dem Beobachter. Die Android-Referenz beschreibt weitere Beobachter und zusätzliche Aspekte des Themas [21].

5.5.3 Menüs im Allgemeinen

Der vorliegende Abschnitt behandelt die Erscheinungsformen und die Definition von Menüs im Allgemeinen. In Android werden zwei Arten von Menüs angeboten:

Optionsmenüs sind Menüs, die über die Menütaste des Geräts aktiviert werden. Pro Activity existiert ein Optionsmenü (Abb. 5-3 links).

Kontextmenüs lassen sich durch längeres Anklicken von Bildschirmelementen (Schaltflächen, Eingabefelder etc.) aktivieren. Für jede View kann ein Kontextmenü definiert werden (Abb. 5-3 rechts).

Richtlinie zur Oberflächengestaltung beachten

Die Richtlinie zur Android-Oberflächengestaltung [38] empfiehlt, Programmfunktionen immer über Schaltflächen oder Menütasten zugänglich zu machen und Kontextmenüs nur als zusätzlichen Weg anzubieten.

Für unsere Anwendung bedeutet das zum Beispiel: Ein Geokontakt soll über das Optionsmenü der Detailansicht gelöscht werden können. Als Zusatzkomfort soll das Löschen ebenfalls über das Kontextmenü der Geokontaktliste angeboten werden.

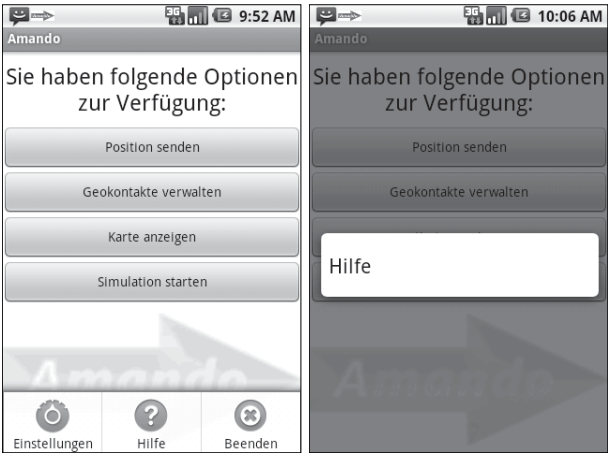


Abb. 5-3
Ein einfaches Optionsmenü (links), einfaches Kontextmenü (rechts)

5.5.4 Menüdefinition

Menüs werden als XML-Ressourcen oder als Java-Code definiert. Wird die Definition per XML durchgeführt, muss dafür eine Datei im Ressourcenverzeichnis `res/menu` angelegt werden. Listing 5.12 zeigt die Definition für das Menü der Amando-Startseite (Abb. 5-3 links). Diese Definition speichern wir als `hauptmenue.xml`-Datei ab. Wir empfehlen, jeweils eine Datei pro Menü und ein Menü pro Activity anzulegen, wobei man den Dateinamen von der Layoutdefinition übernehmen sollte.

Listing 5.12 *Definition des Hauptmenüs von Amando*

```
<menu xmlns:android=
    "http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/opt_einstellungenBearbeiten"
    android:title="@string/men_einstellungenBearbeiten"
    android:icon="@android:drawable/ic_menu_preferences"
  />
  <item
    android:id="@+id/opt_hilfe"
    android:title="@string/men_hilfeAnzeigen"
    android:icon="@android:drawable/ic_menu_help"
  />
  <item
    android:id="@+id/opt_amandoBeenden"
    android:title="@string/men_amandoBeenden"
    android:icon=
        "@android:drawable/ic_menu_close_clear_cancel"
  />
</menu>
```

Diese Definition der Menüinhalte ist sowohl für ein Options- als auch für ein Kontextmenü nutzbar. Pro `<item>`-Element muss das Attribut `android:id` mit dem Wert belegt werden, über den der beschriebene Menüeintrag später ausgewertet werden soll.

Jedes <item> braucht eine Id.

Als Alternative oder Ergänzung der Textdarstellung können für die ersten fünf Einträge eines Optionsmenüs Piktogramme verwendet werden. Man sollte nur kleine, klar unterscheidbare Grafiken einsetzen.

Standard-Menüpiktogramme

Ab Android API 1.5 können die Standard-Menüpiktogramme auf einfache Weise aus der eigenen Anwendung genutzt werden. Der Zugriff erfolgt über den Pfad `android.R.drawable.<icon_resource_id>`. Einige davon sehen Sie hier in den Menüdefinitionen, eine Tabelle aller verfügbaren Piktogramme finden Sie in der Android-Piktogramm-Gestaltungsrichtlinie [22].

Die Tabelle 5-23 gibt eine Übersicht wichtiger Attribute des `<item>`-Elements. Die vollständige Attributliste ist Teil der API-Dokumentation.

Nach dieser allgemeinen Einleitung wollen wir uns die beiden Menüarten genauer ansehen.

Tab. 5-23
Attribute von <item>

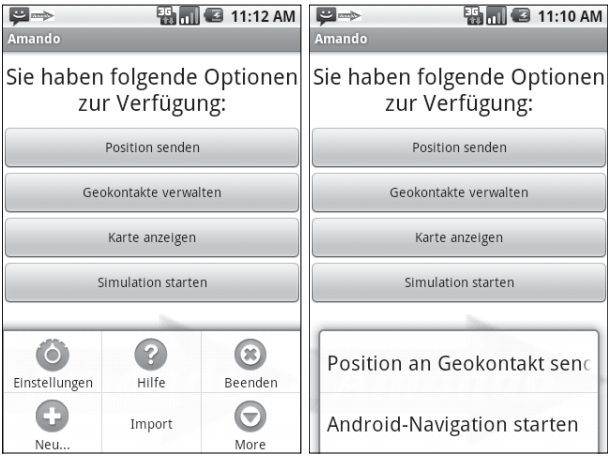
Attribut	Beschreibung	gültig für (Anzahl)
id	Schlüssel des Menüeintrags	Optionsmenüs, Kontextmenüs
title	Langtext Menüoption	Optionsmenüs (6-n), Kontextmenüs
titleCondensed	Abkürzung Menüoption	Optionsmenüs (1-5)
icon	Referenz auf Icon-Ressource	Optionsmenüs (1-5)

5.5.5 Optionsmenüs

Platzprobleme

Jede Activity hat ihr eigenes Optionsmenü. Menüeinträge werden in der Reihenfolge ihrer Definition angezeigt. Falls mehr als fünf Einträge dargestellt werden sollen, wird automatisch ein künstliches »More...[1]«-Element eingefügt, welches die verbleibenden Auswahloptionen als Liste sichtbar macht. Abbildung 5-4 zeigt eine solche lange Optionsliste. Lange Optionsmenüs lassen sich nicht so schnell und komfortabel bedienen wie kürzere. Man sollte daher weitgehend auf sie verzichten.

Abb. 5-4
Optionsmenü mit »More« (links) und ein langes Optionsmenü (rechts)



Zuweisung und Auswertung

Die Definition der Menüeinträge des Optionsmenüs einer Activity findet in der Methode `onCreateOptionsMenu` der Activity statt. Listing 5.13 zeigt den Quelltext für das Hauptmenü in der `Startseite`-Activity. Dabei muss lediglich die Referenz auf die Menü-Ressource angegeben werden. Der Activity Manager des Betriebssystems ruft `onCreateOptionsMenu` nach Erstellung der Activity durch `onCreate` auf.

[1] Der Text hängt von der Spracheinstellung des Geräts ab.

```
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.hauptmenue, menu);
  return super.onCreateOptionsMenu(menu);;
}
```

Listing 5.13 *Zuweisung Optionsmenü*

Hier sollte immer die Methode der Oberklasse aufgerufen werden. Damit kann Android zusätzliche Menüeinträge hinzufügen. Wählt der Anwender eine Menüoption aus, wird die Methode `Activity::onOptionsItemSelected` aufgerufen. Dort wird das gewünschte `<item>` ermittelt und auf die Auswahl reagiert. In Listing 5.14 ist ein Ausschnitt der Implementierung für die `Startseite`-Activity abgedruckt.

```
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    // ... andere Optionen abarbeiten
    case R.id.opt_amandoBeenden:
      finish();
      return true;
    default:
      return super.onOptionsItemSelected(item);
  }
  return false;
}
```

Listing 5.14 *Auswertung Optionsauswahl*

Analog zu den Oberflächenereignissen gibt die Implementierung `true` zurück, wenn das Optionsauswahlereignis verbraucht wurde. Im Standardfall wird an die Oberklasse delegiert.

5.5.6 Kontextmenüs

Für jedes Element einer Bildschirmseite kann ein Kontextmenü definiert werden. Dieses erscheint, sobald der Anwender mit der Auswahltaste länger auf das betroffene Bildschirmelement klickt (*Long-Click Events*). Die Verknüpfung zwischen Kontextmenü und Bezugsobjekt wird in der zugrunde liegenden Activity hergestellt.

> Kontextmenüs werden pro Aufruf neu gefüllt, Optionsmenüs nur einmal bei Erzeugung ihrer Activity.

Es werden immer alle Einträge eines Kontextmenüs angezeigt. Reicht der Platz nicht aus, so wird automatisch eine Bildlaufleiste (engl. *scrollbar*) aktiviert. Das erschwert die Bedienung der Anwendung, so dass man lange Kontextmenüs vermeiden sollte.

Menü zu Kontext

Die Zuweisung eines Kontextmenüs zu einer View erfolgt in zwei Stufen. Als Erstes wird die View bei ihrer Activity für die Verwendung eines Kontextmenüs registriert. Das Listing 5.15 registriert die Liste der `GeoKontaktAuflisten`-Activity mit dem Schlüssel `list`.

Listing 5.15 *Registrierung Kontextmenü für eine View*

```
public void onCreate(Bundle icicle) {
  registerForContextMenu(findViewById(android.R.id.list));
}
```

Damit allein ist das Menü allerdings noch nicht aktiviert. Die Menüoptionen müssen noch zugewiesen werden. Dieser Schritt ist in Listing 5.16 beschrieben. Jede Activity bietet dazu die Methode `onCreateContextMenu` an. Diese ist für die »Befüllung« aller Kontextmenüs der ihr zugewiesenen Bildschirmseite zuständig. `onCreateContextMenu` wird aufgerufen, sobald ein Long-Click Event auf einer für Kontextmenüs registrierten View ausgelöst wurde.

Listing 5.16 *Erzeugung von Kontextmenüs*

```
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {

  if (v.getId() == android.R.id.list) {
    getMenuInflater().inflate(
        R.menu.geokontakt_auflisten_kontext, menu);
  }
  super.onCreateContextMenu(menu, v, menuInfo);
}
```

Das in Amando erscheinende Kontextmenü zeigt die Abbildung 5-5.

Wird nun ein `<item>` dieses Kontextmenüs ausgewählt, so wird vom System die Methode `Activity::onContextItemSelected` aufgerufen. Dort kann auf die gewählte Option reagiert werden (Listing 5.17).

Listing 5.17 *Auswertung eines Kontextmenüs*

```
public boolean onContextItemSelected(MenuItem item) {

  switch (item.getItemId()) {
    case R.id.opt_geokontakt_details: {
      macheEtwasSinnvolles();
      return true;
    }
  }
  return super.onContextItemSelected(item);
}
```

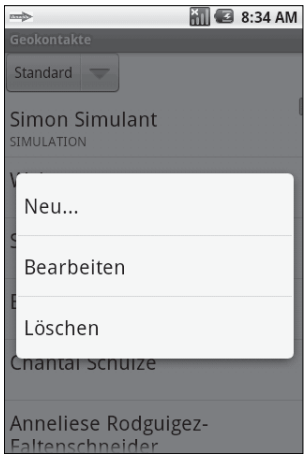


Abb. 5-5
Kontextmenü für Listenelement

Häufig werden wir wissen, auf welche View sich das mitgegebene `MenuItem` bezieht. Hier jedoch haben wir das Menü zwar für eine Liste (`ListView`) registriert, Android erstellt das Kontextmenü aber für einen Listen*eintrag*. Somit stellt sich die Frage, auf welchen Eintrag sich das Kontextmenü bezieht.

Die Antwort liefern die Attribute von `AdapterView.AdapterContextMenuInfo`, in unserem Fall der Schlüssel `AdapterContextMenuInfo::id`. Listing 5.18 zeigt eine neue Version für die Auswertung, die um den nötigen Quelltext ergänzt wurde.

Listing 5.18
Kontextmenü-Auswertung bei einer ListView

```
public boolean onContextItemSelected(MenuItem item) {

  final AdapterView.AdapterContextMenuInfo info =
    (AdapterView.AdapterContextMenuInfo) item.
    getMenuInfo();

  switch (item.getItemId()) {
    case R.id.opt_geokontakt_details: {
      macheEtwasSinnvolles(info.id);
      return true;
    }
  }
  return super.onContextItemSelected(item);
}
```

5.5.7 Dynamische Menügestaltung

In manchen Fällen reicht die statische Definition der Menüs nicht aus. Es wäre wünschenswert, wenn man einzelne Optionen während

des Anwendungslaufs aktivieren, deaktivieren oder komplett verbergen könnte. Die Darstellungsform einzelner `<item>`-Elemente sollte also während des Programmlaufs verändert werden können.

Eine Lösung besteht darin, Menüs nicht als Ressource, sondern direkt im Java-Code zu definieren und zu konfigurieren. Die Android-API sieht dafür die Klasse `android.view.Menu` vor. Das folgende Listing 5.19 aus der `EinstellungenBearbeiten`-Activity zeigt, wie ein Optionsmenü anhand dieser Klasse definiert wird:

Listing 5.19 *Menüdefinition in Java*

```
private static final int EINSTELLUNG_BEARBEITEN_ID =
    Menu.FIRST;
  // ...
public boolean onCreateOptionsMenu(Menu menu) {
  menu.add(0, EINSTELLUNG_BEARBEITEN_ID, Menu.NONE,
      R.string.men_einstellungenBearbeiten);
    // ...
  return super.onCreateOptionsMenu(menu);
}
```

Die Menüdefinitionen über XML und Java schließen sich nicht gegenseitig aus, sondern können nach Belieben kombiniert werden. Anstatt dynamisch per Java Menüeinträge hinzuzufügen, kann zum Beispiel ein Menü-Inflater ein längeres Menü statisch erzeugen, aus dem per Java einzelne Einträge gelöscht werden.

Für die Veränderung eines Optionsmenüs sieht das Android API die Methode `Activity::onPrepareOptionsMenu()` vor. Sie wird vor jeder Anzeige des Optionsmenüs aufgerufen. Listing 5.20 zeigt, wie in der `StartSeite`-Activity die Menüoption für die Hilfe deaktiviert werden könnte.

Listing 5.20 *onPrepareOptions-Menü in Aktion*

```
public boolean onPrepareOptionsMenu(Menu menu) {
  final MenuItem optHilfe = (MenuItem)
    menu.findItem(R.id.opt_hilfe);

  optHilfe.setEnabled(false);

  return super.onPrepareOptionsMenu(menu);
}
```

Indem wir die Menüoption nur deaktivieren, anstatt sie zu löschen, müssen wir sie im umgekehrten Fall nicht erzeugen und hinzufügen. Außerdem folgen wir damit der Empfehlung der Richtlinie zur Android-Oberflächengestaltung [38], die Struktur von Optionsmenüs beizubehalten.

Menüs führen den Nutzer durch die Anwendung. Sie sollten deshalb normalerweise von vornherein klar definiert werden können. Da-

her empfehlen wir, Menüs weitgehend in XML zu definieren und sie nur in Ausnahmefällen im Java-Quelltext zu realisieren.

5.6 Formularverarbeitung

In den letzten Abschnitten haben wir uns mit der Erstellung von Bildschirmseiten beschäftigt. Wir haben Oberflächenelemente auf einer Seite angeordnet und haben diese von einer Activity auf dem Bildschirm darstellen lassen. Wir haben gesehen, wie man verschiedene Arten von Menüs definiert und auf die Berührung einer Schaltfläche reagiert. Was uns jetzt noch fehlt, ist der Zugriff auf View-Elemente.

Zugriff auf Views

In diesem Abschnitt lernen wir, wie man auf die Werte von Views zugreift. Wir sprechen von »*Formularen*«, wenn der Anwender zu einer Eingabe auf dem Bildschirm aufgefordert wird. Für Formulare gibt es verschiedene Views, die Anwendereingaben erlauben. Umgekehrt wird man öfter die Werte in einer View setzen oder austauschen wollen. Auch dann müssen wir im Programm Zugriff auf diese View erhalten und anschließend den Wert setzen.

5.6.1 Zielsetzung

Aufgaben

Wir erstellen in diesem Kapitel eine weitere Bildschirmseite: die Seite zum Anlegen bzw. Bearbeiten eines Geokontakts. Abbildung 5-6 zeigt die Seite im Emulator. Wir erinnern uns, dass wir in der Amando-Anwendung eine eigene Datenbank mit Kontaktdaten verwalten. Zu jedem Kontakt speichern wir dort die letzte bekannte Position, daher nennen wir ihn Geokontakt.

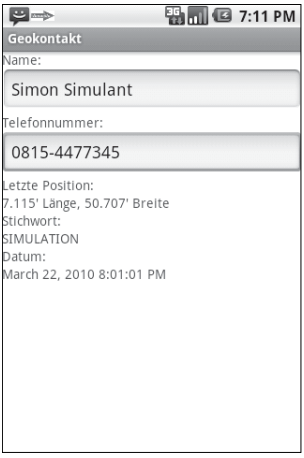


Abb. 5-6
Geokontakt bearbeiten