Android-Wissen

für Technik-Einsteiger und Programmier-Profis

Lernen Sie Ihr Smartpone oder Tablet besser kennen: Apps oder Spiele programmieren und den kleinen grünen Androiden als Freund gewinnen

FRANZIS

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Autoren: Christian Bleske, Manuel Di Cerbo, Michal Gralak, Sven Haiges, Yann Heeser, Christian Immler, Dr. Dirk Koller, Björn Krämer, Christoph Prevezanos, Andreas Itzchak Rehberg, Andreas Rudolf, Torsten Schollmayer, Heike Scholz, René Scholze, Markus Spiering, Thorsten Stark, Patrick Völcker, Roland Willms

ISBN 978-3-645-39076-7

INHALTSÜBERSICHT

Christian Immler

Android-Hacking

Ihr Smartphone kann mehr, als Sie denken:
Hacken Sie Ihr Gerät, bevor es andere tun.

Christian Immler, Jahrgang 1964, war bis 1998 als Dozent für Computer Aided Design an der Fachhochschule Nienburg und an der University of Brighton tätig. Einen Namen hat er sich mit diversen Veröffentlichungen zu Spezialthemen wie 3-D-Visualisierung, PDA-Betriebssystemen, Linux und Windows gemacht. Seit mehr als 20 Jahren arbeitet er als erfolgreicher Autor mit mehr als 200 veröffentlichten Computerbüchern.

Vorwort

Android kann nach gerade einmal sieben Jahren Marktpräsenz – die erste Version erschien am 21.10.2008 – stolz von sich behaupten, das erfolgreichste mobile Betriebssystem aller Zeiten zu sein. Android hat mit etwa 84 % Marktanteil alle anderen mobilen Plattformen weit hinter sich gelassen. Selbst in der Statistik aller Betriebssysteme, nicht nur der mobilen, spielt Android eine wesentliche Rolle. Nach einer Statistik von Statcounter war im September 2015 Windows 7 mit 28,8 % das meistverbreitete Betriebssystem, dicht gefolgt von Android mit 27,5 %. Alle anderen Betriebssysteme erreichten nicht einmal die 10-%-Marke.

Android läuft zurzeit auf über 24.000 verschiedenen Gerätemodellen. Das ursprünglich quelloffen als *Android Open Source Project* AOSP (*source.android.com*) angebotene Betriebssystem wird von den Geräteherstellern mit eigenen Erweiterungen, Oberflächen und Hardwaretreibern angepasst und auch eingeschränkt. Hacken Sie Ihr Smartphone und machen Sie mehr möglich, als der Hersteller zulässt!

Inhaltsverzeichnis

Geheimnisse rund ums »Rooten«

Kurz nach der öffentlichen Vorstellung des ersten Android-Smartphones G1 fand die damals schon sehr aktive Android-Community eine Möglichkeit, einen allumfassenden Systemzugriff auf die Geräte zu erlangen, der dem normalen Benutzer verborgen blieb. Da Android auf Linux basiert, wurde diese Methode als »rooten« bezeichnet, nach dem Linux-Superuser *root*.

Android geht wie jede Linux-Variante von einem normalerweise eingeschränkten Benutzerkonto aus, dem kritische Systemzugriffe verwehrt werden. Ein spezieller Benutzer *root* hat Zugriff auf das komplette System, was natürlich auch mit Risiken verbunden ist. In Android gibt es keinen Standardbenutzer, sondern jede App ist ein eigener Benutzer mit bestimmten Rechten, die bei der Installation durch die in der Datei `AndroidManifest.xml` festgelegten Systemberechtigungen definiert werden und die der Anwender bei der Installation auch bestätigen muss.

»Das Recht auf Root-Zugriff auf Ihr System ist ein zentrales Problem«, sagte Tim Berners-Lee, der Erfinder des World Wide Web auf einer Linux-Konferenz in Australien. *»Das Recht auf Root ist das Recht, Dinge zu speichern, die so laufen, wie Sie es wollen.«* Ein Gerät, das dem Anwender dieses Recht nicht einräume, diene einem fremden Herrn.

Android-Nutzer können sich mit besonderen – von den Geräteherstellern nicht autorisierten – Tools selbst Root-Zugriff auf ihr Smartphone oder Tablet freischalten. Dabei wird der Benutzer *root* angelegt, der normalerweise gar nicht vorhanden ist,

und die Systempartition im Dateisystem komplett mit Schreibzugriff gemountet, damit dieser Benutzer auch schreibend auf alle Dateien zugreifen kann – was diesem Benutzer und allen Apps mit Root-Zugriff verständlicherweise auch zerstörerische Fähigkeiten ermöglicht.


Sind Sie sich nicht sicher, ob Ihr Smartphone gerootet ist oder nicht, hilft die kostenlose App *Root Verifier* weiter. Diese App überprüft das Smartphone auf mögliche Root-Zugriffe, ohne es selbst zu rooten oder irgendwelche anderen Veränderungen vorzunehmen. Damit der Test auf einen möglichen Root-Zugriff funktioniert, fragt die App wie jede App, die Root-Rechte benötigt, bei der Superuser-App nach. Hier muss eine entsprechende Anfrage bestätigt werden, um den Test durchzuführen.

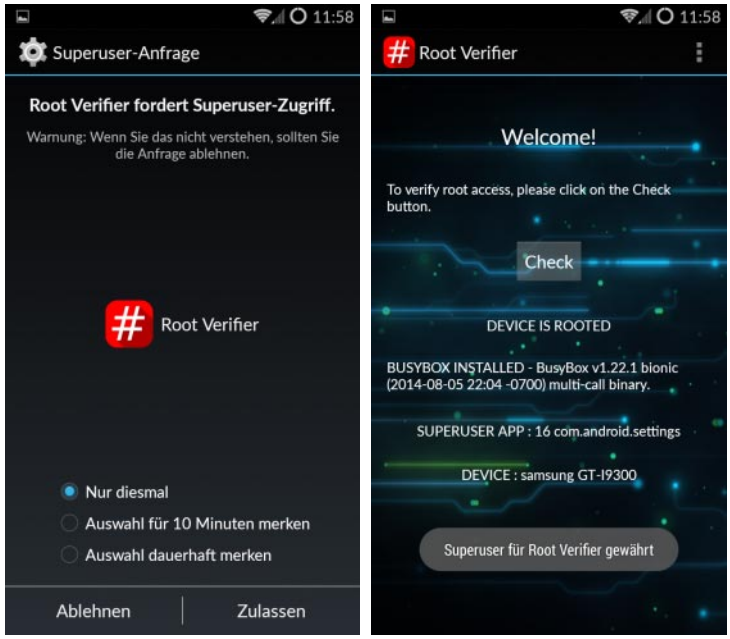


Bild 1.1: Root Verifier stellt fest, ob ein Smartphone gerootet ist oder nicht.

Ist auf dem Smartphone kein Google Play Store vorhanden, weil zum Beispiel ein CustomROM installiert ist, kann Root Verifier auch über den F-Droid-App-Store heruntergeladen werden.

Das #-Zeichen, das die Root Verifier-App als Logo nutzt, ist übrigens das Linux-interne Symbol für den Root-Benutzer und wird im Zusammenhang mit Rooten noch öfter auftauchen.

Rooten wird häufig dafür genutzt, nicht benötigte Bloatware-Apps, die Gerätehersteller oder Mobilfunkprovider auf den Smartphones vorinstalliert haben, zu ent-

fernen, was bei in der Firmware installierten Apps auf klassischem Weg nicht möglich ist.

Was bedeutet Firmware?
Klassischerweise unterscheidet man in der Computertechnik zwischen Hardware und Software. Hardware ist die Elektronik, und Software sind die Programme, einschließlich Betriebssystem, die diese Elektronik steuern. Firmware liegt irgendwo dazwischen. Dabei handelt es sich um Programme oder Betriebssysteme, die fest auf einem Chip eingebrannt sind und deren Zweck darin besteht, die Grundlage zu liefern, damit weitere Software die Hardware steuern kann. Da auch Firmware in letzter Zeit immer wieder vom Hersteller aktualisiert wird, ist sie heute nicht mehr fest in einem Chip eingebrannt, und es gibt Methoden, sie zu verändern. Die dazu notwendigen Werkzeuge waren ursprünglich nur den Herstellern zugänglich. Die Android-Community und auch Google selbst stellen mittlerweile diverse derartige Tools zur Verfügung.

Im gerooteten Modus sind noch weit mehr Funktionen möglich, was auch Entwicklern einen großen Spielraum für spezielle Root-Apps bietet. So ist zum Beispiel das Übertakten des Prozessors möglich, da bestimmte Schutzmechanismen des Systems übergangen werden müssen. Das Übertakten bringt zwar mehr Leistung, geht aber zulasten des Stromverbrauchs und der Lebensdauer von Prozessor und Akku. Umgekehrt kann ein Smartphone auch untertaktet werden. Es läuft dann langsamer, was in den meisten Apps – außer Spielen – gar nicht auffällt. Dafür wird weniger Strom verbraucht, und die Hardware wird geschont.

Zur Installation sogenannter CustomROMs, wie unter anderem CyanogenMod, ist nicht zwingend Root-Berechtigung nötig. Es reicht ein entsperrter Bootloader aus. Allerdings werden häufig ROM-Manager-Apps zur Installation von CustomROMs verwendet. Sie benötigen Root-Berechtigung zur Installation eines Recovery.

Ist Rooten gefährlich?
Wenn man nicht weiß, was man tut: Ja! Einige schlecht informierte Sensationsmedien suggerieren unbedarften Android-Nutzern, Rooten sei das einzig Wahre. Für den Normalanwender ist genau das Gegenteil der Fall. Ihnen bringt Rooten wesentlich mehr Risiken als Vorteile. Erst durch Rooten werden Android-Smartphones wirklich gefährdet, was Viren und andere Malware betrifft, die üblicherweise durch systemeigene Sicherheitsfunktionen geblockt werden.
Technisch entspricht das Rooten dem Einrichten eines Administratorbenutzers auf einem PC, der volle Rechte auf das gesamte System hat und damit – auch versehentlich – jeden beliebigen Schaden anrichten kann.

Das Rooten an sich ist zwar mittlerweile weitgehend sicher, danach kann man durch Fehlbedienung oder bösartige Apps, die ohne Root-Zugriff nicht laufen, sein System eventuell unwiderruflich beschädigen. Im Gegensatz zu einem durch Fehlbedienung beschädigten Windows-PC lässt sich das Betriebssystem auf einem Android-Smartphone nicht so einfach neu installieren. Es gibt standardmäßig auch keine Systemwiederherstellung und keine Rettungs-CDs. Da beim Rooten alle Sicherheitsmechanismen außer Kraft gesetzt werden, kann angebliche Tuning-Software, die Prozessor oder Grafikchips übertaktet, diese auch tatsächlich hardwareseitig beschädigen oder gar zerstören.
CustomROMs benötigen zur Installation Root-Zugriff, der danach aber bei vielen ROMs, wie zum Beispiel bei CyanogenMod, automatisch wieder eingeschränkt wird und vom Benutzer für bestimmte Apps explizit freigeschaltet werden kann. Mit einem Mindestmaß an Wachsamkeit kann man ohne Weiteres verhindern, dass zweifelhafte Apps den Root-Zugriff bösartig ausnutzen.

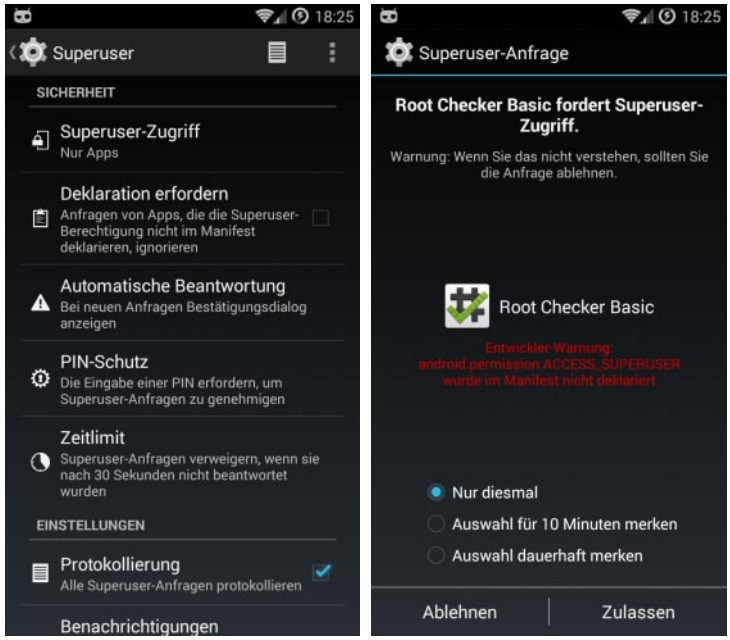


Bild 1.2: Einstellungen für Root-Zugriffe in CyanogenMod und Anfrage einer App auf Root-Zugriff.

Ist Rooten illegal?
Nein! Im Gegensatz zu Jailbreaks auf dem iPhone werden durch das Rooten keine Urheberrechte verletzt und auch keine bewusst gesetzten Sperren aufgebrochen. Die meisten Hersteller lehnen Garantieansprüche für gerootete Geräte aber grundsätzlich ab. Die Free Software Foundation Europe *fsfe.org* vertritt dagegen den Standpunkt, dass das Rooten nach der Richtlinie 1999/44/CE des Europäischen Parlaments die zweijährige Garantie auf die Hardware nicht verletzt, solange der Händler nicht beweisen kann, dass ein Defekt durch das Rooten entstanden ist, wie dies z. B. bei Übertaktung der Fall wäre.

Android 5 Lollipop überprüft vor einem automatischen OTA-Firmwareupdate den Zustand des Systems. Ist das Gerät gerootet, ist in den meisten Fällen kein OTA-Firmwareupdate möglich.

Rooten und SIM-Lock
Ein eventuell auf dem Smartphone vorhandener SIM-Lock wird durch das Rooten nicht automatisch entfernt. Allerdings benötigen die meisten Apps zum Aushebeln von SIM-Locks Root-Zugriff.

1.1 Rooten – so geht's

Es gibt grundsätzlich drei verschiedene Methoden, ein Smartphone zu rooten:

- Rooten mit einer App
- Rooten mit einer PC-Software über die USB-Kabelverbindung
- Rooten durch Überspielen einer speziellen gerooteten Firmware auf das Smartphone

Das Rooten mit einer App, die sich direkt auf dem Smartphone Root-Rechte verschafft, ist theoretisch die einfachste Variante. Diese Apps nutzen Sicherheitslücken in einzelnen Android-Versionen, die aber schnell wieder weitgehend geschlossen werden, um zu verhindern, dass Apps sich unbemerkt Root-Rechte verschaffen und damit das System beschädigen oder Sperren umgehen, um Malware einzuschleusen. Diese Tools müssen immer wieder aktualisiert werden und funktionieren nur so lange mit einzelnen Betriebssystemversionen, bis Google oder der Gerätehersteller einen Sicherheitspatch liefert.

Rooting-Tools, die vom PC aus arbeiten, benötigen eine USB-Kabelverbindung zum Smartphone und können folglich gar nicht unbemerkt laufen. Das Smartphone muss dazu im Entwicklermodus sein, und USB-Debugging muss eingeschaltet sein. Auch dazu muss der Entwickler bewusst eingreifen, Apps haben keine Möglichkeit, ein Smartphone unbemerkt in den Entwicklermodus zu versetzen.

Rooten rückgängig machen
Da das Rooten eine reine Softwaremaßnahme ist, lässt es sich auch wieder rückgängig machen, solange Sie nicht im gerooteten Zustand ein anderes ROM installiert haben. Die meisten Rooting-Tools verfügen sogar über eine Unroot-Funktion. Im Fall eines Garantieanspruchs kann ein Gerätehersteller aber durchaus erkennen, ob ein Fehler durch Rooten hervorgerufen wurde.

1.2 Rooten – Vorbereitung und Grundlagen

Als Erstes brauchen Sie zum Rooten, wie auch später zum Flashen von Custom-ROMs, ein paar Tools auf dem PC, die Google über das Android-SDK für Entwickler, und für jeden anderen, kostenlos zur Verfügung stellt. In vielen Fällen reicht auch das weiter unten beschriebene Paket *Minimal ADB and Fastboot*.

Apps aus unbekannten Quellen zulassen
Damit während des Rootens auf dem Gerät auch die notwendigen Apps installiert werden können, muss in den Einstellungen unter *Sicherheit* der Schalter *Installation von Apps aus unbekannten Quellen zulassen* eingeschaltet sein.

1.2.1 Das Android-SDK

Laden Sie sich bei *developer.android.com/sdk* das Android-SDK herunter. Sie benötigen die einfachen *Stand-alone SDK Tools*, nicht das *Android Studio*. Letzteres ist nur für Entwickler, die eigene Apps für Android entwickeln wollen.

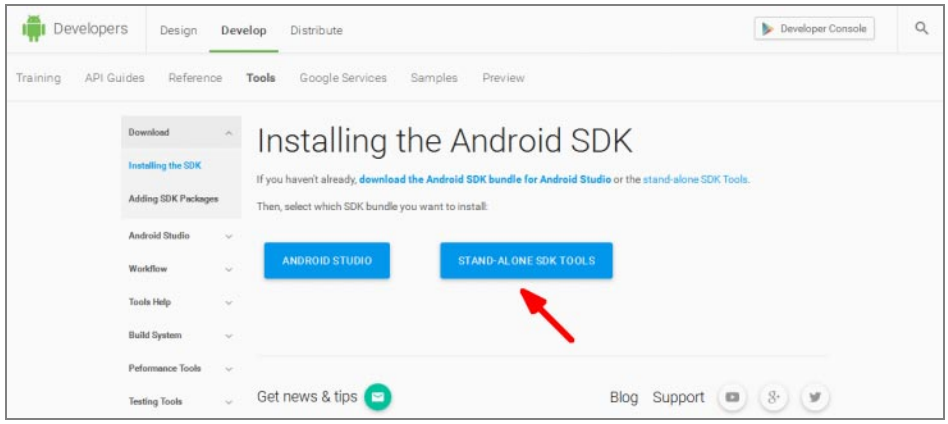


Bild 1.3: Stand-alone SDK Tools installieren.

Zur Installation des Android-SDK benötigen Sie auf dem PC das Java SE Development Kit (JDK). Sollte dies nicht vorhanden sein, kann es automatisch aus dem Android-SDK-Installer nachinstalliert werden.

Starten Sie aus dem Hauptverzeichnis des Android-SDK das Programm `SDK Manager.exe`. Hier werden jede Menge Pakete zum Download und zur Installation angeboten, von denen Sie aber nur die wenigsten brauchen.

Sie brauchen aus dem Bereich *Tools* die *Android SDK Tools* und *Android SDK Platform-tools*. Aus dem Bereich *Extras* brauchen Sie die *Google USB Driver* und *Usb Driver*. Die diversen sehr großen Betriebssystemabbilder für SDK und Emulator benötigen Sie

nicht. Wählen Sie die benötigten Komponenten aus und klicken Sie auf *Install packages*. Die Pakete werden heruntergeladen und im SDK installiert.

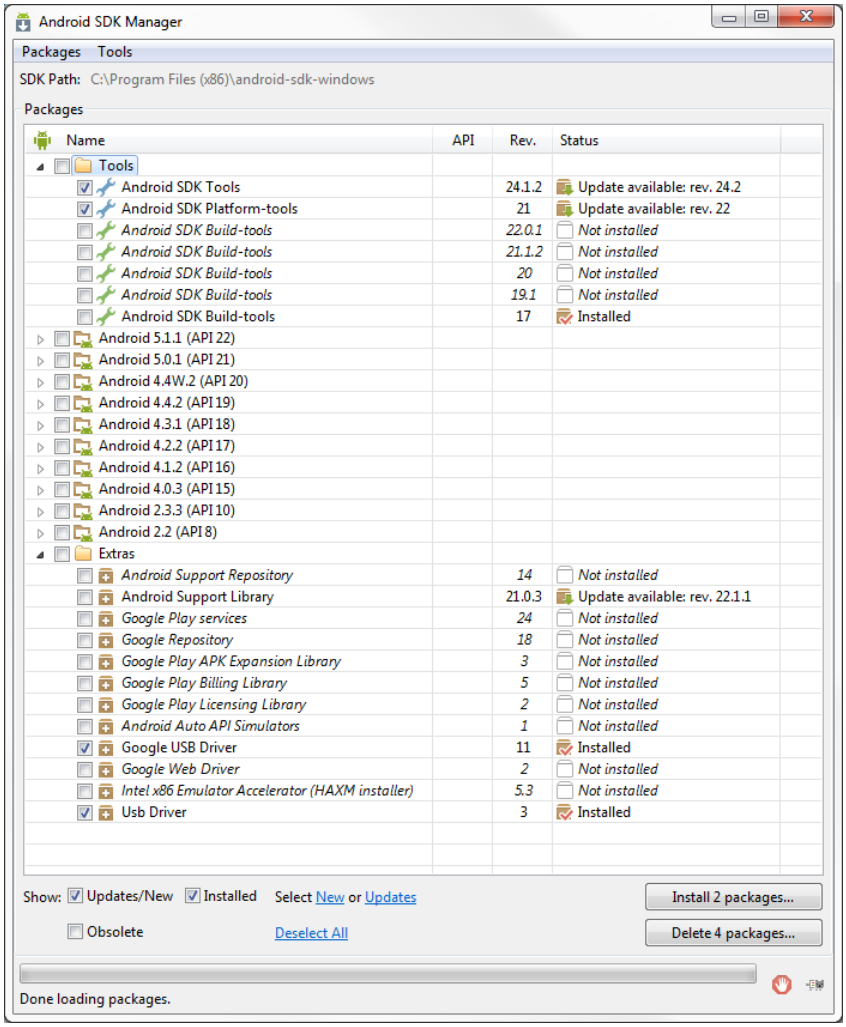


Bild 1.4: Der Android SDK Manager.

1.2.2 Minimal ADB and Fastboot

Wer nicht weiter in die Tiefe von Android vordringen, sondern nur CyanogenMod oder ein anderes CustomROM installieren möchte, benötigt auf den meisten Smartphones nicht das komplette Android-SDK. Die Tools aus dem Paket *Minimal*

ADB and Fastboot reichen meistens aus. Sie finden dieses Paket zum Download unter *forum.xda-developers.com/showthread.php?t=2317790* (*bit.ly/1FaUwEC*).

Das xda-developers Forum

Das xda-developers Forum ist eine große weltweite Community von Android-Entwicklern. In letzter Zeit finden sich dort auch zunehmend Windows Phone-Entwickler. Viele private Entwickler nutzen dieses Forum als »offizielles« Download- und Informationsportal für ihre Software. Die dort angebotene Software ist in der Regel als seriös einzustufen. Da es sich um ein Entwicklerportal handelt, wird natürlich das entsprechende technische Verständnis im Umgang mit den Tools vorausgesetzt.

Wie immer bei Software gilt auch bei Smartphone-Firmware:

Googeln Sie nie nach einer Datei – schon gar nicht zusammen mit dem Wort Download!

Hersteller von Malware setzen alles daran, in den Suchergebnissen bei Google ganz oben zu stehen. Gerade bei systemkritischen Aktionen wie dem Flashen einer Smartphone-Firmware haben es unseriöse Entwickler leicht, unbedarfte Anwender davon zu überzeugen, Malware bei ausgeschalteten Sicherheitsmechanismen auf ihre Geräte zu überspielen.

Das Toolpaket *Minimal ADB and Fastboot* taucht bei Google auch auf diversen mehr oder weniger seriösen Downloadportalen auf, die es in eigene Bloatware-Installer verpacken, mit denen Sie sich neben dem eigentlichen nur etwa 2 MByte großen Programmpaket jede Menge »Tod und Teufel« auf den PC holen.

Installieren Sie das heruntergeladene Paket auf dem PC. Da es sich bei allen Tools im Paket um Kommandozeilentools handelt, öffnet der Startmenüeintrag nur ein Eingabeaufforderungsfenster, das automatisch ins Verzeichnis der Tools springt.

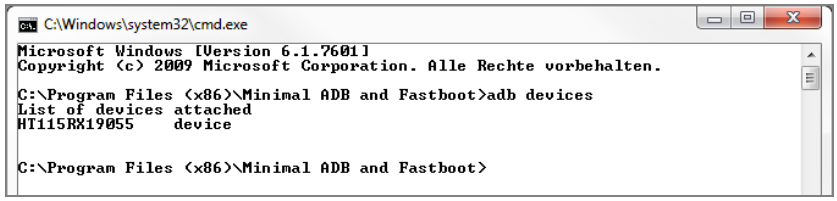


Bild 1.5: Kommandozeilentool Minimal ADB and Fastboot.

Falls Sie auf dem PC spezielle Software zur Datenübertragung zum Smartphone verwenden, wie zum Beispiel *Samsung Kies* oder *HTC Sync*, deinstallieren Sie sie jetzt. Die Programme nur auszuschalten reicht nicht aus, da die bei derartigen Tools mitgelieferten Spezialtreiber ebenfalls deinstalliert werden müssen.

Christian Immler

CyanogenMod
Installation und Praxis

Christian Immler, Jahrgang 1964, war bis 1998 als Dozent für Computer Aided Design an der Fachhochschule Nienburg und an der University of Brighton tätig. Einen Namen hat er sich mit diversen Veröffentlichungen zu Spezialthemen wie 3-D-Visualisierung, PDA-Betriebssysteme, Linux und Windows gemacht. Seit mehr als 15 Jahren arbeitet er als erfolgreicher Autor mit mehr als 20 veröffentlichten Computerbüchern.

INHALTSVERZEICHNIS

1 CYANOGENMOD – DAS BESSERE ANDROID

Android ist grundsätzlich ein quelloffenes und freies Betriebssystem für Smartphones und Tablets, das jeder seinen Wünschen anpassen kann ... theoretisch. In der Praxis sieht es anders aus. Im Wesentlichen ist es Google, der größte und wichtigste Android-Entwickler, der vorgibt, was Android kann und was nicht. Anschließend verpassen die Smartphone-Hersteller dem Betriebssystem eigene Oberflächen, die das System oft ausbremsen und Funktionseinschränkungen mit sich bringen.

Eine unabhängige Entwicklergruppe hat auf Basis von Android das verbesserte Open-Source-Betriebssystem *CyanogenMod* herausgebracht und entwickelt es ständig weiter. CyanogenMod wurde von jeglichem Ballast wie Spyware (Software, die Benutzer ausspioniert) und Bloatware (automatisch vorinstallierte Werbe-Apps unterschiedlichster Anbieter – Onlineshops, Lieferdienste, Taxi, Hotels ...) befreit und enthält zusätzliche Systemfunktionen sowie erweiterte Einstellungen, die die offiziell auf den meisten Smartphones installierten Android-Versionen nicht bieten. Dadurch wurden die allgemeine Geschwindigkeit und auch die Zuverlässigkeit verbessert.

ANDROID 6.0 MARSHMALLOW

Seit Ende des Jahres 2015 steht die neue CyanogenMod-Version 13 für die ersten Geräte zur Verfügung. Sie basiert auf Android 6 Marshmallow. Die wenigsten Gerätehersteller liefern bisher offizielle Marshmallow-Updates für bereits früher ausgelieferte Smartphones. CyanogenMod ermöglicht jetzt schon einem großen Nutzerkreis, in den Genuss der neuesten Android-Version zu gelangen, und dies sogar auf Smartphones, die von ihren Herstellern möglicherweise nie ein Update erhalten werden.

Die meisten Abbildungen in diesem Buch wurden mit CyanogenMod 13 gemacht. Allerdings gilt das meiste auch für CyanogenMod 12, da die Unterschiede zwischen Android 5.x Lollipop und Android 6.0 Marshmallow bei Weitem nicht so gravierend sind, wie in den Medien gern behauptet.

Bild 1.1: Auf den ersten Blick sieht CyanogenMod wie klassisches Android ohne Google-Dienste aus.

Als Benutzer finden Sie sich in CyanogenMod sofort zurecht, da die Oberfläche und alle wichtigen Symbole direkt aus Android übernommen wurden. Die Oberfläche wirkt schlanker, da die übliche Bloatware fehlt, die neuen Optionen sind an verschiedenen Stellen, thematisch passend, integriert.

1.1 Warum CyanogenMod?

Warum sollte man auf seinem Smartphone CyanogenMod installieren, wo es doch ein vorinstalliertes Android gibt, das (meistens) auch funktioniert? Eine berechtigte Frage, die sich viele stellen werden.

- **Aktuelle Android-Versionen** werden von Geräteherstellern oft nur mit erheblichen Zeitverzögerungen oder gar nicht geliefert. Oft sind es produktpolitische und gar nicht technische Gründe, die Gerätehersteller dazu bewegen, ein Smartphone auf dem Stand einer bestimmten Android-Version stehen zu lassen und nicht mehr mit Updates zu versorgen. CyanogenMod liefert selbst für

Smartphones, die schon einige Jahre vom Markt verschwunden sind, noch aktuelle Android-Versionen.

- CyanogenMod beseitigt **Bloatware**, die sich auf klassischem Weg nicht deinstallieren lässt.
- Durch den Verzicht auf unnötige Hintergrunddienste läuft CyanogenMod selbst auf schwächeren Geräten flüssiger und liefert so eine deutlich **bessere Performance** als die mit den Geräten gelieferte Android-Version.
- Die **Benutzeroberfläche** ist schlanker und vielfältiger anpassbar. Einige dieser Funktionen sind im Standard-Android über externe Launcher ebenfalls möglich.
- **Zusätzliche Systemfunktionen**, die durch die eingeschränkten Benutzerrechte standardmäßig nicht möglich sind, werden in CyanogenMod mit einem kontrolliert einsetzbaren Root-Zugriff möglich gemacht.
- Für manche Anwender mag das Gefühl wichtig sein, die **Kontrolle über das eigene Smartphone** zu haben, da sich CyanogenMod auch ohne Google-Dienste verwenden lässt.

CYANOGENMOD IN ZAHLEN

CyanogenMod ist mit über 50 Millionen Installationen weltweit das beliebteste aller CustomROMs für Android und war im Jahr 2009 auch eines der ersten, die überhaupt veröffentlicht wurden. CyanogenMod hatte Anfang August 2015 bereits mehr Nutzer als die großen »offiziellen« Plattformen Windows Phone und BlackBerry zusammen. Es läuft zurzeit auf 351 (Stand: März 2016) verschiedenen offiziell unterstützten Gerätemodellen sowie etwa 200 weiteren nur inoffiziell unterstützten Modellen.

1.2 Die wichtigsten Zusatzfunktionen in Kürze

Einige Funktionen in CyanogenMod liegen im Hintergrund und nutzen vor allem technisch interessierten Anwendern. Die folgende Liste zeigt die augenfälligen Zusatzfunktionen, die jedem Anwender einen Mehrwert bieten. Allerdings stehen nicht alle davon auf jedem Gerät und in jeder CyanogenMod-Version zur Verfügung.

- Design über Themen – und nicht nur Hintergrundbilder – weitreichend anpassbar.
- Schnelleinstellungen als Band- oder Kachelansicht einstellbar.

- Dateimanager vorinstalliert – fehlt im Standard-Android leider immer noch.
- Profile für verschiedene Situationen oder Umgebungen, war früher Standard auf jedem Handy, ging aber mit Android verloren.
- Erweiterte Einstellungen für Startbildschirm und Benachrichtigungen.
- Erweiterte Kontrolle darüber, welche App Zugriff auf persönliche Daten haben darf.
- Blockierliste für unerwünschte Anrufer.
- Anzahl der zu versendenden SMS begrenzen.
- Tasten des Smartphones frei belegbar.
- Schnellzugriffe für wichtige Apps auf dem Sperrbildschirm.
- Integrierter Musikplayer funktioniert auch komplett offline ohne Zwangsanbindung an einen Musikstore.
- Systemweit nutzbarer Equalizer.
- Root-Zugriff für bestimmte Apps kontrolliert einsetzbar.

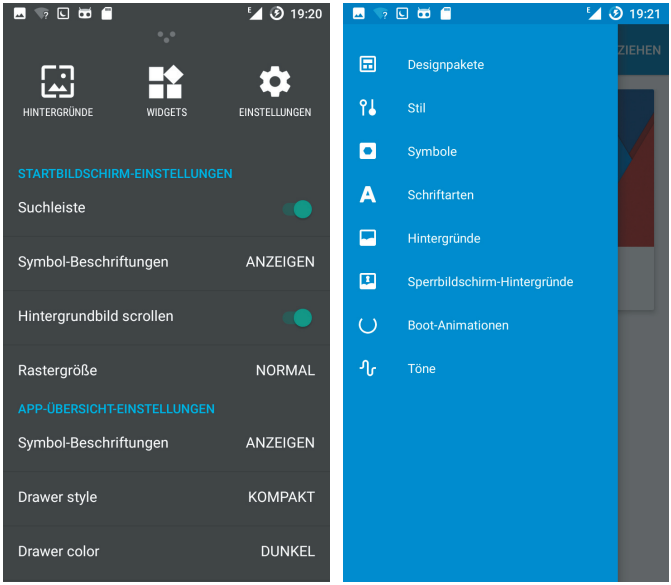


Bild 1.2: Einstellungen für Startbildschirm und Designs.

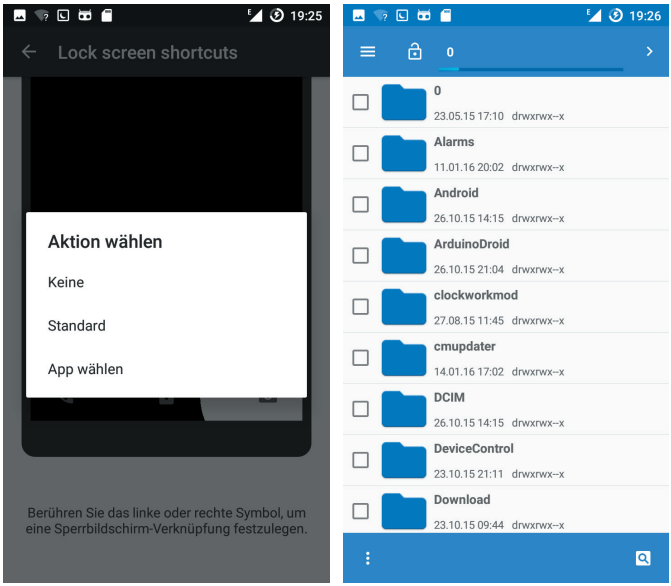


Bild 1.3: Wichtige Apps auf den Sperrbildschirm und in den Dateimanager holen.

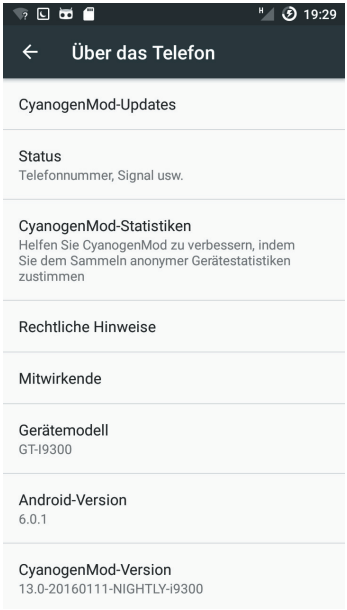
1.3 Die unterschiedlichen CyanogenMod-Versionen

CyanogenMod wurde erstmals auf Basis der Version Android 1.5 Donut veröffentlicht und seitdem ständig weiterentwickelt. Android Donut war die erste Version, die auf einem im Markt erhältlichen Gerät – dem T-Mobile G1 – vorinstalliert war.

Die folgende Tabelle zeigt die Versionsnummern von CyanogenMod und die jeweils dahinterliegenden Android-Versionen. Zu jeder CyanogenMod-Version gibt es Unterversionen, die wichtige Schritte in der permanent stattfindenden Weiterentwicklung markieren. Nicht alle CyanogenMod-Versionen sind für alle Smartphones verfügbar.

Version 12 ist für die meisten halbwegs aktuellen Geräte die aktuelle stabile CyanogenMod-Version, Version 13 steht seit Ende Januar 2016 als stabile Version für die ersten Geräte zur Verfügung, aber auch ältere Hauptversionen werden weiterentwickelt, um zusätzliche Geräte zu unterstützen oder Fehler zu beseitigen.

CYANOGENMOD-VERSION	ANDROID-VERSION	JAHR DER ERST-VERÖFFENTLICHUNG
3	1.5 Cupcake	2009
4	1.6 Donut	2009
5	2.0/2.1 Eclair	2010
6	2.2 Froyo	2010
7	2.3 Gingerbread	2011
8 (Entwicklung eingestellt)	3.0 Honeycomb	2011
9	4.0 Ice Cream Sandwich	2012
10	4.1/4.2/4.3 Jelly Bean	2012
11	4.4 KitKat	2013
12	5.0/5.1 Lollipop	2015
13	6.0 Marshmallow	2015



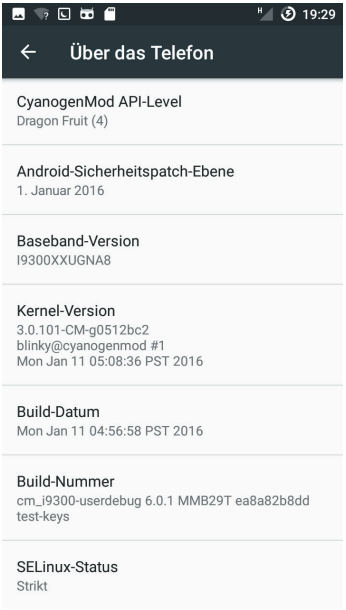


Bild 1.4: Versionsnummern und Daten in den Einstellungen anzeigen.

Die installierte CyanogenMod-Version wie auch den Gerätenamen und die Android-Version finden Sie wie üblich in den Einstellungen unter *Über das Telefon*.

Android 6 Marshmallow zeigt in der Zeile *Android-Sicherheitspatch-Ebene* das Datum der aktuellsten Sicherheitsupdates an. Auf diese Weise lässt sich der aktuelle Updatestand bei neu aufgetauchten Sicherheitslücken feststellen, da nicht jedes kleine Update bereits eine höhere Versionsnummer trägt. Google plant, etwa monatlich aktuelle Sicherheitsupdates für Android 6 Marshmallow zur Verfügung zu stellen. CyanogenMod zeigt zusätzlich das Build-Datum an, an dem die installierte Version erstellt wurde.

CYANOGEN OS

Neben dem quelloffenen CyanogenMod liefert der Hersteller Cyanogen Inc. noch eine kommerzielle Variante seines Betriebssystems an Gerätehersteller aus, die dieses anstelle von Google Android auf ihren Smartphones vorinstallieren können. Cyanogen OS (*cyngn.com/cyanogen-os*) ist auf dem sehr populären Smartphone OnePlus One und mittlerweile auf weiteren Geräten installiert, die vor allem im asiatischen Raum verkauft werden. In Europa wird das Aquaris X5 des spanischen Herstellers BQ mit Cyanogen OS ausgeliefert.
Im Funktionsumfang gleichen sich Cyanogen OS und CyanogenMod weitgehend. Im Rahmen einer neuen Partnerschaft mit Microsoft wird Cyanogen OS in Zukunft Microsoft-Apps und -Dienste enthalten, unter anderem OneDrive, Office, Skype und Bing. Cyanogen Inc. plant sogar eine komplett Google-freie Variante seines Betriebssystems, in der Cortana, die digitale Assistentin aus Windows 10, anstelle der Google-Suche und Google-Sprachsteuerung eingesetzt werden soll.

Bild 1.5: Smartphones mit Cyanogen OS: OnePlus One, BQ Aquaris X5, YU Yuphoria, YU Yureka, Alcatel Onetouch Hero 2+.

Andreas Itzchak Rehberg

Das inoffizielle Android-Handbuch

4., aktualisierte und erweiterte Auflage

Einsteiger-Workshop, Apps, Datensicherung, Sicherheit,
Privatsphäre, Tuning, Root-Zugang und mehr:
Mit Android können Sie mehr als nur telefonieren!

Vorwort

Bei diesem Buch handelt es sich um eine Übersicht, die den Einstieg in den Umgang mit einem Android-Gerät erleichtern soll. Im hinteren Teil werden aber auch viele Inhalte für Fortgeschrittene geboten.

Eine wichtige Grundlage für die Inhalte sind meine App-Reviews nach Einsatzzweck bei AndroidPIT (*www.androidpit.de*), die ich hier sinnvoll zusammenzufassen versucht habe. Viele der Links in diesem Buch, die über QR-Codes eingebunden sind, führen daher auch dorthin – zur Vertiefung eines Themas etwa oder für weitere Details und nicht zuletzt für aktualisierte Informationen: Es kommen ja ständig neue Apps hinzu und natürlich ebenso wertvolle Benutzererfahrungen. Darüber hinaus lassen sich im Forum Fragen zum Buch stellen (und Antworten erwarten), auch zu hier nicht behandelten Themen.

App-Reviews nach Einsatzzweck
bit.ly/LSy66d

Gedacht ist das Ganze so, dass dieses Buch einen Überblick verschafft. Für tiefschürfendere Dinge kann man auf den ebenfalls bei Franzis erschienenen zweiten Band, »Das inoffizielle Android System-Handbuch«, aber auch auf das Forum zurückgreifen. Dort ist man nicht nur in Bezug auf Android-Fragen in guten Händen!

Und noch etwas muss ich loswerden: Viele der hier kurz vorgestellten (oder auch nur genannten) Apps habe ich nicht selbst getestet – beispielsweise weil ich nicht die Voraussetzungen dazu habe (ich nutze kein Facebook und meinen Androiden auch nicht zum Spielen, um nur zwei Dinge zu nennen). Trotzdem habe ich sie – der Vollständigkeit halber – beschrieben und greife dabei auch auf Erfahrungen der Anwender in der Community zurück, die diese Apps benutzen.

Hinweise zur Benutzung des Buchs

Am Seitenrand finden sich hin und wieder die schon erwähnten QR-Codes. Sie sollen helfen, den Anschluss ins Internet zu finden: Sind sie schwarz, führen sie zu weiteren Informationen, die in Grau leiten direkt weiter zur besprochenen App.

Um die QR-Codes nutzen zu können, braucht es zwei Zutaten: ein Android-Gerät mit integrierter Kamera und eine App, die sich auf QR-Codes versteht. Für Letztere gibt es weiter unten im Buch einige Hinweise (in Kapitel 5.9, »Büro, Office & Verwaltung«). Ersteres hat der Leser dieses Buchs in der Regel bereits (falls nicht, stehen die zugehörigen Adressen zusätzlich unter den Codes, damit man auch mit anderen Geräten an die Infos herankommt).

Wie werden diese Codes genutzt? Ganz einfach: Androiden zücken, den QR-Codereader starten und die Kamera auf den QR-Code richten. Um alles Weitere kümmert sich die entsprechende App dann selbstständig: Sie öffnet in der Regel den Webbrowser und ruft die durch den Code festgelegte Webseite auf. Ganz bequem also.

Dass sich mit diesen Codes noch Weiteres anstellen ließe und was alles – diese Informationen befinden sich in Kapitel 5.9.1, »Barcodes«.

Was ist bit.ly?
Damit man keine langen URLs abtippen muss, verwenden wir teilweise den Abkürzungsdienst `bit.ly`. Dieser setzt lange Originallinks in deutlich kürzere um. Man tippt z. B. `bit.ly/1kS9qXI` in die Adresszeile des Browsers ein (dabei die Groß- und Kleinschreibung beachten) und wird automatisch zu `www.franzis.de/smartphone-multimedia/das-inoffizielle-android-system-handbuch` weitergeleitet. Die QR-Codes verweisen generell direkt auf die originale URL.
URL-Abkürzungsdienste wie `bit.ly` kommen in den Medien immer wieder wegen Sicherheitsrisiken ins Gerede, da auf den ersten Blick für den Nutzer nicht zu erkennen ist, wo ein verkürzter Link hinführt. Wer bei einem bit.ly-Link Sicherheitsbedenken hat, fügt am Ende ein +-Zeichen an, zum Beispiel: `bit.ly/1kS9qXI+`. dann wird eine Informationsseite mit dem tatsächlichen Link angezeigt, statt dass direkt auf die jeweilige Website weitergeschaltet wird.

Einige der hier vorgestellten Tools und Apps setzen den root-Zugang (vgl. Abschnitt 6.1) zum Gerät voraus. Stellen, an denen solche Apps beschrieben werden, sind am Seitenrand mit einem Symbol und einem grauen Balken markiert.

Danksagung

Ja, hört der denn mit der Vorrede gar nicht mehr auf? Gleich, gleich. Aber dieser Abschnitt muss noch sein:

Denn bedanken muss ich mich auf jeden Fall. Nicht nur, weil sich das eben so gehört – sondern weil ich dazu viele gute Gründe habe. Ohne den Rückhalt der Community bei AndroidPIT wäre es nie zu diesem Buch gekommen! Und so bedanke ich mich besonders herzlich bei Evelyn für ihre tatkräftige Unterstützung und Hilfe (was hätte ich nur ohne dich gemacht?) und Sabine für ihr fleißiges Gegenlesen und Aufspüren von »Leichen«. (Leider verschwand doch die eine oder andere App wieder aus dem Play Store, bevor ich das Buch fertig hatte. Zum Glück ist so etwas nicht die Regel!) Auch Alexander möchte ich für seine zahlreichen Hinweise danken. Und all den anderen, die ich hier jetzt nicht alle namentlich aufführen kann: Leute, ihr seid klasse!

Und von der Community gesprochen: Auch die M&Ms waren mehr als nur hilfreich. M&Ms? Nun ja, die Moderatoren und die »Macher«. Mein besonderer Dank geht hier an Mario, Michael, Fabien und Philipp.

android.
stackexchange.
com

Viele Ideen stammen auch von StackExchange (*http://android.stackexchange.com/*). Hier gilt mein besonderer Dank ce4, Flow, Ryan, t0mm13b, Liam und ganz speziell Dan Hulme – aber auch der gesamten Community: Macht echt Spaß bei euch! Und man lernt jeden Tag etwas Neues dazu!

Ein ganz besonderer Dank geht an meine Frau, die schon glaubt, ich wäre mit dem Computer zusammengewachsen. Zum Glück brachte sie statt einer Säge oder eines Tranchiermessers Nervennahrung an meinen Schreibtisch. Danke für deine Geduld mit mir!

Abschließend noch meinen Dank an die Leser dieses Buchs, die es bis hierhin durchgehalten haben und die hoffentlich auch noch ein wenig weiterlesen: viel Spaß bei der Lektüre!

Inhaltsverzeichnis

2

Mit Android arbeiten

2.1 Steuerzentrale: Einstellungen und Switches

Haben wir den Homescreen als »Schaltzentrale« bezeichnet, ist der Ort, an dem die Systemeinstellungen getätigt werden, wohl die »Steuerzentrale«. Und es gibt so einiges einzustellen bei Android, die Liste ist also nicht unbedingt kurz. Hinzu kommt, dass vieles »historisch gewachsen« ist – und somit manche Dinge an den verschiedensten Orten zu suchen sind, obwohl sie aus subjektiver Sicht eigentlich zusammengehören.

Klar, es handelt sich bei aktuellen Android-Versionen schon um recht komplexe Systeme, in denen man an vielen Schräubchen drehen können muss. Doch insbesondere für Einsteiger sind das meist zu viele (wobei genau die, die man gern hätte, natürlich fehlen). Doch gibt es auch hier einige Apps, die für Erleichterung sorgen: entweder weil sie die Auswahl auf wesentliche (häufig genutzte) Punkte zusammenstauchen oder weil sie in spezifischen Bereichen zusätzliche Einstellungsmöglichkeiten schaffen.

2.2 Konfiguration

Bei Android lässt sich so einiges konfigurieren. Und mit jeder neuen Version kommen neue Dinge hinzu. Ich möchte nicht auf alles eingehen – doch einige zentrale Einstel-

lungen sollten hier erläutert werden. Die folgenden Dinge sind alle in den *Einstellungen* von Android untergebracht. Wie man dorthin gelangt? Vom Homescreen ausgegangen, geht es zunächst über die Menütaste ins Menü und von hier zum Punkt *Einstellungen*. Dann geht es so weiter, wie in den folgenden Abschnitten beschrieben.

2.2.1 WLAN

Klar, mit so einem Smartphone möchte man gern ins Netz. Und wenn man noch keinen vernünftigen Datentarif gebucht hat: Was liegt näher, als das heimische WLAN zu nutzen? Oder das bei Freunden und Verwandten? Zumal es in der Regel ja auch schneller ist als die mobile Datenverbindung. Was also ist zu tun?

In den *Einstellungen* wählen wir den Punkt *Drahtlos & Netzwerke*. Hier lässt sich WLAN generell aktivieren (indem man das passende Häkchen setzt bzw. den entsprechenden Schalter »umlegt«). Sodann tauchen wir in den Punkt *WLAN-Einstellungen* ab – und gelangen zu einem Bildschirm, der dem hier dargestellten ähnelt.

Bild 2.1: Die WLAN-Einstellungen.

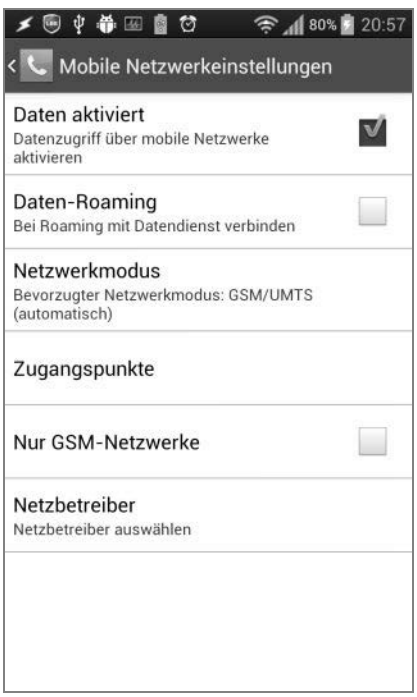


Bild 2.2: Die Konfigurationsmöglichkeiten für das mobile Netz.

Der erste Punkt entspricht dem generellen Aktivieren der WLAN-Funktion (wie auf der vorherigen Seite). Ist WLAN aktiv und mit einem Netzwerk verbunden, wird das an dieser Stelle auch angezeigt. Mit dem zweiten Punkt kann man sich »unterwegs« über verfügbare offene WLAN-Netzwerke informieren lassen. Wer jedoch vertrauliche

Daten auf seinem Androiden hat, sollte mit solchen Netzen vorsichtig sein (siehe Kapitel 3.9, »In fremden Netzen«): Man weiß ja nie …

Darunter werden nun alle aktuell verfügbaren WLAN-Netze aufgelistet. Ebenfalls dargestellt wird, ob (und wenn ja wie) sie verschlüsselt sind. Hier müsste also auch das »eigene« WLAN (bzw. das, in das man sich einbuchen will) nun stehen. Einfach antippen, gegebenenfalls den Schlüssel (das »Verbindungspasswort«) eingeben und auf *Verbinden* tippen – wenige Sekunden später sollte die Verbindung stehen. Bei Bedarf lässt sich ab Android 4.0 nach Aktivieren der Checkbox *Erweiterte Optionen einblenden* auch ein für das jeweilige Netzwerk zuständiger Proxyserver angeben.

Hat man sie einmal eingegeben, merkt sich Android die Verbindungsdaten übrigens: Kommt man das nächste Mal bei aktiviertem WLAN in die Nähe dieses Netzes, erfolgt die Verbindung automatisch.

2.2.2 Mobiles Datennetz

Der »moderne Mensch« ist ja heutzutage permanent online. Unser WLAN können wir aber nicht überallhin mitnehmen. Was tun?

Die Antwort heißt: einen passenden Datentarif (Volumentarif oder Flatrate) buchen und das »mobile Datennetz« konfigurieren! Ersteres gibt es beim Provider – und Letzteres findet sich wieder unter *Drahtlos & Netzwerke* (ab Android 4.0 versteckt es sich hinter *Mehr*). Der Punkt *Mobilfunknetze* (bzw. *Mobile Netzwerke*) führt dann zu einem Bildschirm ähnlich Bild 2.2.

Ein kurzer Blick auf das Bild dürfte auch bereits helfen, eine der größten Sorgen auszuräumen: Was ist mit meinen Datenkosten, wenn ich im Ausland bin? Ja, was? Das hängt ganz davon ab, was auf dieser Seite konfiguriert wurde. Standardmäßig sind die Häkchen bei *Roaming* nicht gesetzt (also wie auf dem Bild zu sehen). Im Ausland bzw. generell im Netz eines Fremdanbieters wird daher die Datenverbindung gar nicht erst aufgebaut. Also sollten hier diesbezüglich auch keine Kosten entstehen. Diesem (und möglichen Alternativen) ist ein eigenes Kapitel (2.3, »Roamingkosten vermeiden«) gewidmet. Weiter unten lässt sich noch ein Häkchen setzen, das die Datenverbindung auf »2G« beschränkt (*Nur GSM-Netzwerke*). Das ist zwar nicht so schnell wie »3G« oder gar »4G«, spart aber unter Umständen einiges an Energie (siehe 6.5.6, »2G versus 3G: Spart 2G wirklich so viel Akku?«), sodass man mit einer Akkuladung länger auskommt. Für ein wenig E-Mail und Web ist das auch völlig ausreichend. Sollte man tatsächlich einmal mehr Durchsatz benötigen, kann jederzeit umgeschaltet werden.

www.
cyanogenmod.
com

Woher weiß Android denn nun, wie es ins Internet kommt? Diese Einstellungen verbergen sich hinter den *Zugangspunkten*. So manch Custom-ROM (wie z. B. CyanogenMod) ermittelt diese Konfiguration automatisch: Anhand der SIM-Karte erkennt es den Anbieter und ordnet die entsprechenden Zugangsdaten aus seiner Datenbank zu.

2.2.3 Tethering


Wireless Tether

An dieser Stelle folgt oft die Frage: »Ich habe da noch ein Tablet/ Notebook/... Kann ich irgendwie die mobile Datenverbindung mit nutzen?« Vor Android 2.2 (aka Froyo oder »Frozen Yoghurt«) hieß die Antwort eindeutig: Jein. Mit root und einer App wie *Wireless Tether* ließ sich das erreichen. Ansonsten galt der übliche Spruch: »Ohne root sich nichts tut!«

Zum Glück hat sich das eindeutig geändert: Seit Froyo gehört Tethering zur »Standardausrüstung«, und es ist natürlich auch noch bei den aktuellen Versionen mit an Bord (siehe Screenshot). Wer sichergehen möchte, dass kein Dritter »mitsurft«, kann das Netzwerk über USB weiterreichen. Einfacher geht es jedoch, wenn man seinen Androiden in einen »mobilen Hotspot« umwandelt. Hierzu wird der zweite im Bild gezeigte Punkt aktiviert, und die Details werden unter *WLAN-Hotspot-Einstellungen* eingetragen: Eine SSID (Name für den Zugangspunkt) kann nach Gusto vergeben, eine Verschlüsselung gewählt und natürlich auch der zugehörige Schlüssel bzw. das Passwort hinterlegt werden. Und schon steht dem Surfvergnügen nichts mehr im Wege ...

Wer immer noch ein wenig unsicher ist, hat vielleicht bemerkt: Da gibt es einen Punkt namens *Hilfe*. Stimmt. Und da wird das Ganze auch noch mal erklärt – falls dieses Buch gerade mal nicht zur Hand ist ...

Je nach Hersteller, Gerät, Android-Version und ROM sieht dieses Menü allerdings unterschiedlich aus: Einmal fehlt *USB-Tethering* (zum Beispiel bei meinem LG Optimus 4X mit Android 4.0.3), oder es sind zusätzlich Dinge verfügbar wie *WiFi Direct* oder *NFC*. Also bitte nicht wundern.

2.2.4 Internettelefonie

Bild 2.3: Android bietet ab Version 2.2 (Froyo) Tethering »ab Werk«.

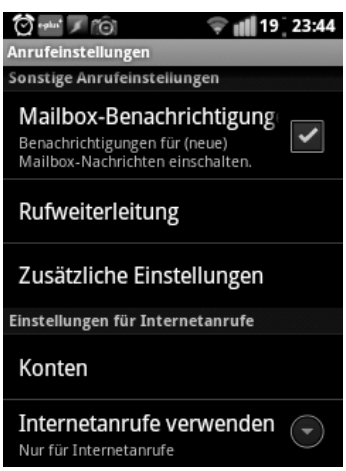


Bild 2.4: Die Anrufeinstellungen.

Michal Gralak/Thorsten Stark

Schnelleinstieg App Usability

Plattformübergreifendes Design:
Android, Apple iOS und Windows Phone

Michal Gralak entwickelt bereits seit seinem Medieninformatik-Studium an der Beuth Hochschule Berlin Apps für iOS-Systeme. Nachdem er in Berührung mit Hardware-Accessories für iPhones kam, flammte seine Begeisterung für Bluetooth-gestützte Apps auf. Als Software Engineer bei BURY Technologies entwickelte er einzigartige Apps für die hauseigene Freisprecheinrichtung sowie innovative Apps für die VW-Tochter Bentley. Mittlerweile entwickelt er als Produktmanager bei BURY Technologies neue Produkte für den Automotive- und Aftermarketsektor und nutzt dafür sein langjähriges Wissen um die Einbindung von mobilen Endgeräten in den Business- und Lifestyle-Bereich.

Thorsten Stark hat an der Beuth Hochschule für Technik in Berlin (damals noch TFH Berlin) seinen Abschluss als Diplominformatiker sowie den Master of Science im Studiengang Medieninformatik gemacht. Derzeit arbeitet er an seiner Promotion, in der er sich mit der Usability von Tablets beschäftigt.

In fünf Jahren in Forschungsprojekten hat er viel praktische und theoretische Erfahrung in der Gestaltung von Nutzerschnittstellen gesammelt. Dabei lag das Hauptaugenmerk immer im mobilen Bereich, aber auch andere Multi-Touch-Geräte kamen nicht zu kurz. Als iOS-Entwickler der ersten Stunde kann er auf eine langjährige Erfahrung im mobilen Bereich zurückblicken und hat auch die Entwicklung vom ersten iPhone bis heute mitverfolgt.

Danksagung

Viele Personen haben rund um das Buch an der Entstehung mitgewirkt und durch viel Einsatz sowie Feedback zum Endergebnis beigetragen. Diesen Personen würden wir gerne namentlich noch einmal Danke sagen.

▶ Michal Gralak dankt:

Onur Güngören, meinem Co-Founder und erkek kardeş. Danke für die Unterstützung und deine unendliche Geduld in der gesamten Zeit.

Frau Prof. Ilse Schmiedecke, die mich während meines Studiums für Usability sensibilisiert hat und deren Vorlesungen sehr hilfreich für die Entstehung dieses Buches waren.

Markus Stäuble, der mich geduldig immer wieder angetrieben und im Schreiben bestärkt hat.

Jennifer Möller, die gerade in der Endphase des Buches sehr viel Geduld und Verständnis aufweisen musste.

Katharina Thies für ihre Geduld, die guten Tipps, Hinweise zum psychologischen Teil des Buches und vieles mehr.

Thorsten Stark, der sich bereit, erklärt hat, dieses Projekt mit mir gemeinsam zu stemmen – durch dich habe ich viel gelernt!

Meiner Mutter für ihre langjährige und hilfreiche Unterstützung, ihre Geduld und überhaupt für alles!

▶ Thorsten Stark dankt:

Herrn Prof. Dr. Manfred Thüring, meinem Doktorvater, für den tiefen Einblick in das Thema und die Unterstützung.

Frau Prof. Dr. Gudrun Görlitz dafür, dass sie mir die Möglichkeit gegeben hat, mich intensiver mit dem Thema zu beschäftigen, und die Unterstützung bei der Promotion.

Michal Gralak dafür, dass er mir überhaupt erst die Gelegenheit gegeben hat, an diesem Buch mitzuarbeiten – ich denke, wir haben beide viel voneinander gelernt, auch wenn (oder gerade weil) wir nicht immer derselben Meinung waren.

Janine Riethbaum für die liebe Unterstützung und Geduld in all der Zeit. Ich liebe dich.

Meinen Kollegen (Alex, Damian, Dominik, Hannes, Jessica und Mark), denen ich regelmäßig mit dem Thema Usability bei ihren Projekten auf die Nerven gehe.

Und natürlich meinen Eltern, die mich mein ganzes Leben begleitet und immerzu unterstützt haben und für mich da waren.

Inhaltsverzeichnis

2 Hardwarevoraussetzungen

Checkliste – Apps auf Mobilgeräten bedienen

- ☐ Kleine Bildschirme.
- ☐ Nach Möglichkeit mit einer Hand bedienbar.
- ☐ Sinnvolle Umschaltung zwischen Hoch- und Querformat je nach Haltung des Geräts.
- ☐ Unterschiedliche Lichtverhältnisse bei App-Gestaltung berücksichtigen.
- ☐ Internetverbindung kann ausfallen.

Was fällt denn eigentlich genau in die Kategorie der Mobilgeräte? Ist ein Laptop auch mobil? Für dieses Buch definieren wir Mobilgeräte als Geräte, die klein und leicht sind, (fast) immer mitgeführt werden, ausschließlich über einen Touchscreen bedient werden und persönlich sind – kurz gesagt, Smartphones, Tablets und Phablets (Smartphones mit Bildschirmgrößen von 6 Zoll und mehr).

2.1 Bildschirmgrößen und -auflösungen

Wenn wir attraktive Apps aufbauen, funktioniert das zu einem großen Teil über die grafische Bedienoberfläche. Um diese Oberfläche ansprechend zu gestalten, ist es für Entwickler wichtig, zu wissen, welche Größe, welches Format und welche Auflösung der Bildschirm hat, für den man entwickelt. Schließlich muss die Schrift lesbar sein, sie darf aber nicht so viel Platz einnehmen, dass andere Gestaltungselemente verschwinden. Gutes Design hängt also maßgeblich davon ab, wie gut man die Zielplattform kennt.

Was ist, wenn sich die Größe des Bildschirms ändern kann? Dann müssen sich auch alle Elemente der Benutzeroberfläche und ihre Abstände an die neuen Dimensionen anpassen.

Trotz der Vielzahl von Auflösungen haben sich die Hersteller der mobilen Betriebssysteme etwas einfallen lassen, um den Entwicklern entgegenzukommen. Bei iOS ist es beispielsweise egal, ob man für ein iPad mit Retina-Display oder ohne entwickelt. Das interne Koordinatensystem, mit dem man als Entwickler arbeitet, ist in beiden Fällen 1.024 × 768. Lediglich bei der Darstellung später wird die höhere Auflösung verwendet, um ein schärferes Bild zu erhalten.

Retina-Display

Apple nennt seine Bildschirme mit hoher Dichte an Pixeln werbewirksam Retina-Display. Damit meinen sie, dass man mit bloßem Auge und bei normalem Abstand zwischen Augen und Display keine Pixel mehr unterscheiden kann. Dass das MacBook Pro mit Retina-Display lediglich die halbe Dichte des iPhones hat, wird damit begründet, dass der Bildschirm des Laptops in der Regel weiter vom Gesicht weg ist als ein Smartphone.

Unter Android gibt es das Konzept der Density-Independent-Pixel (zu Deutsch: auflösungsunabhängige Pixel), kurz: dp. Dabei wird zwischen unterschiedlichen Pixeldichten unterschieden: *ldpi* (geringe Pixeldichte), *mdpi*, *hdpi* und *xhdpi* (hohe Pixeldichte) sowie *xxhdpi* und *xxxhdpi* (sehr hohe Pixeldichte). Dadurch kann der Entwickler sämtliche Größen- und Positionsangaben in dp machen, und je nachdem, auf welchem Gerät die App läuft, werden die Angaben in echte Pixel umgerechnet.

Bildschirmgrößen beim Smartphone

Sieht man sich die im Folgenden abgebildeten Auflösungen an, fällt auf, dass sie stark variieren. Das betrifft nicht nur die Größe, sondern auch das Seitenverhältnis.

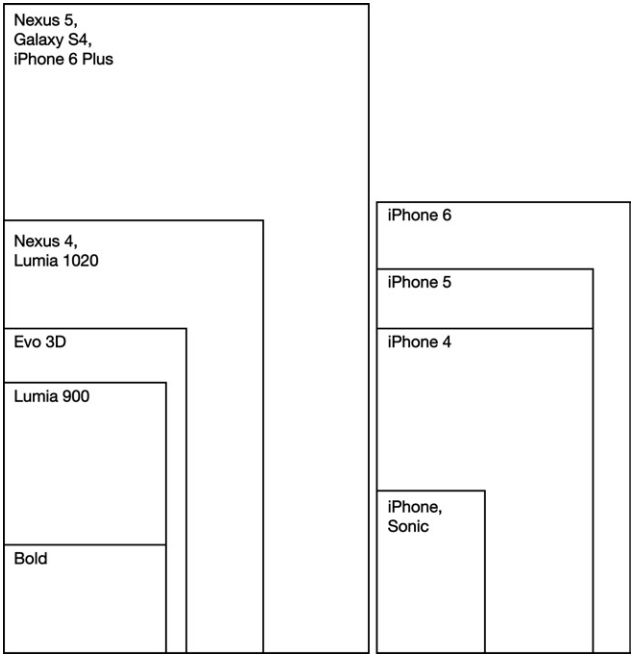


Bild 2.1: Unterschiedliche Auflösungen bei Smartphones.

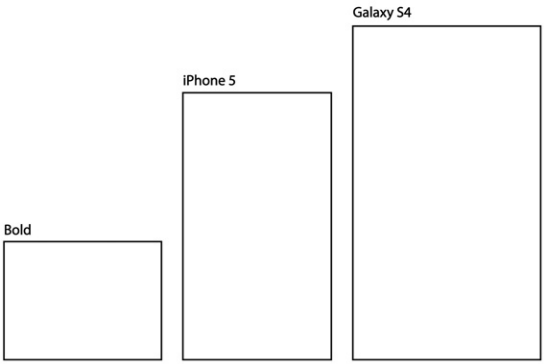


Bild 2.2: Größenvergleich der physischen Displays.

Name	Diagonale (Zoll)	Auflösung (Pixel)	Pixeldichte (ppi)	Maße (cm)	Jahr
BlackBerry Bold	2,8	480 × 320	206	5,2 × 3,9	2008
Apple iPhone	3,5	320 × 480	163	4,9 × 7,4	2007
Apple iPhone 4	3,5	640 × 960	326	4,9 × 7,4	2010
Apple iPhone 5	4	640 × 1.136	326	4,9 × 8,8	2012
Google Nexus 4	4,7	768 × 1.280	318	5,9 × 9,8	2012
Google Nexus 5	5	1.080 × 1.920	445	6,2 × 11	2013
Nokia Lumia 900	4,3	480 × 800	216	5,6 × 9,4	2012
Nokia Lumia 1020	4,5	768 × 1.280	331	5,9 × 9,8	2013
Huawei Sonic	3,5	320 × 480	163	4,9 × 7,4	2011
Samsung Galaxy S4	4,99	1.080 × 1.920	441	6,2 × 11	2013
HTC Evo 3D	4,3	540 × 960	256	5,3 × 9,5	2011

Tabelle: Smartphone-Screens in Zahlen.

Nachdem Apple es fünf Jahre lang geschafft hat, die physische Bildschirmgröße des iPhones beizubehalten, haben sie sich 2012 dafür entschieden, ihre neuen iPhones mit größeren 4-Zoll-Bildschirmen – anstelle der alten 3,5 Zoll – auszustatten. Das iPhone 5 hat zwei Jahre später einen physisch größeren Bildschirm erhalten. Die Breite wurde beibehalten, lediglich in der Länge sind einige Pixel dazugekommen. Das interne Koordinatensystem wurde entsprechend angepasst: von 320 × 480 auf 320 × 568.

Für iOS-Entwickler hat sich also seit den Anfängen die Bildschirmbreite nicht verändert. Programmtechnisch waren es immer 320 Points. Das erleichtert die Gestaltung der Benutzeroberfläche, denn man kann sich sicher sein, dass das Layout identisch aussehen wird, egal auf welchem Gerät die App letztendlich läuft. Nur die Höhe kann bei den neuen Modellen etwas größer ausfallen als bei den vorherigen. Dies lässt sich aber recht einfach bewerkstelligen.

Mit dem iPhone 6 hat Apple erstmals in der Geschichte der iPhones den Bildschirm deutlich verändert. Die Geräte sind sichtbar größer als die Vorgänger.

Apple bietet mit Auto Layout eine Möglichkeit, die grafische Oberfläche einer App dynamisch an unterschiedliche Gegebenheiten anzupassen. Dazu ist nicht einmal eine Zeile Code nötig, denn Auto Layout ist in den Interface Builder von Xcode integriert. Man kann darüber nicht nur die seitlichen Abstände festlegen, sondern auch Abhängigkeiten herstellen. Objekte können aneinander ausgerichtet werden, und die Abstände zwischen einzelnen Objekten können definiert werden, z. B. »≤10 Points«. Dann wird Auto Layout versuchen, diese so anzupassen, dass ihr Abstand maximal 10 Points oder weniger beträgt. Wir reden hier von Points und nicht von Pixeln. Pixel sind die physischen Bildpunkte im Screen, Points hingegen die »virtuellen Pixel« des logischen Koordinatensystems. Das Verhältnis von Point zu Pixel beträgt je nach Endgerät 1 : 1 (nicht Retina) ,1 : 2 (Retina) oder 1 : 3 (iPhone 6 Plus).

Mit den Density-Independent-Pixeln bei Android verhält es sich ähnlich wie mit dem internen Koordinatensystem unter iOS. Als Entwickler hat man ein normalisiertes Koordinatensystem, mit dem man arbeitet, und je nach Gerät wird es dann später mit einem entsprechenden Faktor multipliziert, z. B. mit 0,75 bei einem ldpi-Bildschirm oder mit 1,5 bei einem hdpi-Bildschirm. Da die Hersteller jedoch immer mehr höher auflösende Bildschirme verbauen, werden ldpi, mdpi und hdpi immer seltener verwendet.

Tablet

Smartphones haben Bildschirmgrößen bis ca. 6 Zoll, Tablets hingegen fangen meistens erst bei 7 Zoll an – nach oben hin offen. Jedoch gibt es kaum Tablets, die größer als 13 Zoll sind. Während kleinere Tablets (<10 Zoll) wegen ihrer Größe gern unterwegs genutzt werden, ist vielen Menschen ein Tablet größer als 10 Zoll dafür zu unhandlich und zu schwer. Diese haben ihren Einsatz eher auf dem heimischen Sofa oder in Meetings. Dort werden sie in erster Linie zum Konsumieren aller Arten von Medien verwendet.

Bei Tablets ist die Vielfalt von Auflösungen nicht ganz so hoch wie bei Smartphones. Die meisten bewegen sich in Größenordnungen zwischen 1.024 × 600 und 2.560 × 1.600 Pixeln. Das Nexus 7 beispielsweise hat eine sehr hohe Pixeldichte und eignet sich wegen seines länglichen Bildschirms eher zum Lesen von Büchern oder Betrachten von Filmen. Texte zu schreiben ist im Querformat eher anstrengend, da die virtuelle Tastatur schon die Hälfte des Bildschirms einnimmt und nach Abzug von Menü- und Statusleiste kaum noch Platz bleibt für den Text. Beim iPad hinge-

gen hat man mehr Platz für den Text, dafür kann es passieren, dass sich der Text hinter der Tastatur versteckt. Das passiert besonders dann, wenn der Benutzer die geteilte Tastatur gewählt hat. Diese befindet sich dann nicht mehr am unteren Rand, sondern beliebig weiter oben. Das wäre ergonomisch betrachtet sogar ganz gut, jedoch passen die Schreibprogramme das virtuelle Papier nicht daran an, und es verteilt sich auf den kompletten Platz zwischen oberer und unterer Kante. Schade nur, dass da mitten drauf jetzt eine Tastatur liegt.

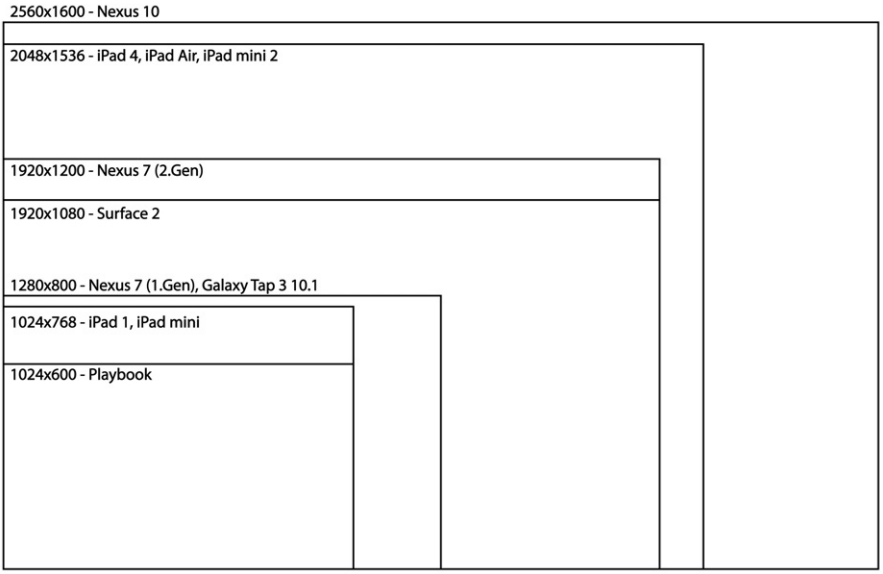


Bild 2.3: Unterschiedliche Auflösungen bei Tablets.

Name	Diagonale (Zoll)	Auflösung (Pixel)	Pixeldichte (ppi)	Maße (cm)	Jahr
BlackBerry Playbook	7	1.024 × 600	170	15,3 × 9	2010
Apple iPad 1	9,7	1.024 × 768	132	19,7 × 14,8	2010
Apple iPad 4	9,7	2.048 × 1.536	264	19,7 × 14,8	2012
Apple iPad Air	9,7	2.048 × 1.536	264	19,7 × 14,8	2013

Name	Diagonale (Zoll)	Auflösung (Pixel)	Pixeldichte (ppi)	Maße (cm)	Jahr
Apple iPad mini	7,9	1.024 × 768	163	16 × 12	2012
Apple iPad mini 2	7,9	2.048 × 1.536	326	16 × 12	2013
Google Nexus 7 1.Gen.	7	1.280 × 800	216	15 × 9,4	2012
Google Nexus 7 2.Gen.	7	1.920 × 1.200	323	15 × 9,4	2013
Google Nexus 10	10	2.560 × 1.600	300	21,6 × 13,5	2012
Samsung Galaxy Tab 3 10.1	10,1	1.280 × 800	149	21,7 × 13,6	2013
Microsoft Surface 2	10,6	1.920 × 1.080	207	23,5 × 13,2	2013

Tabelle: Tablet-Screens in Zahlen.

Phablet

Die dritte Gerätekategorie sind die Phablets, manchmal auch Smartlets genannt. Der Name setzt sich – wie Sie wahrscheinlich schon vermutet haben – aus den Begriffen »Smartphone« und »Tablet« zusammen. Damit ist der Name nur die logische Konsequenz dessen, was die Geräte sind – nämlich eine Mischung aus Smartphone und Tablet. Man kann sie ebenso gut als große Smartphones oder kleine Tablets mit Telefonfunktion beschreiben. Ihre Anzahl ist noch überschaubar, aber der Trend geht immer mehr in diese Richtung. Verfolgt man die Entwicklung der iPhones oder der Samsung Galaxy S-Reihe, fällt auf, dass sie in den letzten Jahren immer größer geworden sind. Von 3,5 auf 5,5 Zoll bei den iPhones und von 4 auf 5 Zoll bei den Galaxy S.

Die wohl bekanntesten Vertreter der Phablets sind die Geräte der Samsung-Galaxy-Note-Reihe. Ihr 5,3 bis 5,7 Zoll großer Touchscreen kann zusätzlich mit einem induktiven Stift – S-Pen genannt – bedient werden. Damit sind handschriftliche Notizen, Handschrifterkennung und Zeichnen direkt auf dem Bildschirm möglich.

2.2 Multithreading

Der Trend zu 64-Bit-CPUs mit mehreren Kernen, wie bei PCs schon seit einigen Jahren üblich, hält nun auch in die mobile Welt Einzug.

Mehrere Kerne bedeuten auch mehrere Threads. Threads sind Abläufe, die parallel stattfinden können. Dies passiert aber nicht automatisch. Zumindest für die Teile des Codes, die wir selbst schreiben, ist es daher sinnvoll, bestimmte Aufgaben in separate Threads auszulagern. Diese können dann auf einem weiteren Kern der CPU bearbeitet werden oder, falls es sich um eine Single-Core-CPU handelt, zumindest im Wechsel mit den anderen Threads. Das ist immer dann sinnvoll, wenn Abläufe länger dauern können. Würde man diese im selben Thread abarbeiten wie den Rest der App, würde die App einfrieren und so lange nicht bedient werden können, bis die Operation abgeschlossen ist. Der Main-Thread, in dem unsere App läuft, ist auch zugleich der UI-Thread, also verantwortlich dafür, Eingaben entgegenzunehmen und mit entsprechenden Ausgaben zu reagieren. Und wenn die App nicht auf Eingaben reagieren kann, weil sie gerade mit etwas anderem beschäftigt ist, denkt der Benutzer, die App sei abgestürzt. Im besten Fall wartet er dann einfach etwas, bis sich auf dem Display wieder was tut, im schlimmsten Fall beendet er jedoch die App und löscht sie sofort.

Daher sollten Sie als Entwickler dafür Sorge tragen, dass die App jederzeit auf Eingaben reagieren kann. Falls wirklich einmal längere Berechnungen oder Dateioperationen anstehen, sollte dies dem Benutzer auf jeden Fall mitgeteilt werden – am besten durch eine Fortschrittsanzeige, denn dann kann der Benutzer auch abschätzen, wie lange er ungefähr noch warten muss. Wenn man aus irgendwelchen Gründen den Fortschritt nicht ermitteln kann, tut es z. B. unter iOS auch der *Activity-Indikator* (Aktivitätsanzeige). Das ist dieser Kreis aus kleinen Strichen, der oben neben dem Netzwerksymbol angezeigt wird oder prominent in der Bildschirmmitte erscheint. Damit weiß der Benutzer, dass die App noch läuft, aber gerade mit anderen Aufgaben beschäftigt ist und daher nicht auf Eingaben reagieren kann.

Bild 2.4: Activity-Indikator in der Mail-App unter iOS 7.

2.3 Netzwerk/Mobilfunk

Das ist das Schöne heutzutage: Wir haben mit unseren Computern, Smartphones, Tablets und Laptops überall und jederzeit Zugang zum Internet. Theoretisch zumindest – praktisch nein. In dem Moment, in dem wir anfangen, uns zu bewegen, ist das nicht mehr der Fall. In Bürogebäuden gibt es häufig flächendeckendes WLAN. Außerhalb ist man jedoch auf das Mobilfunknetz oder freie WLAN-Hotspots angewiesen. Und genau dort kommt es immer wieder zu Schwankungen und Abbrüchen in der Verbindung.

Wenn möglich, sollten Sie versuchen, Ihre App so zu gestalten, dass sie auch ohne Netzwerkverbindung arbeiten kann.

Versuchen Sie, den Netzwerkverkehr auf ein Minimum zu beschränken, denn er kostet Zeit und den Benutzer unter Umständen auch Geld. Mobile Datenflatrates sind meistens auf ein bestimmtes Datenvolumen beschränkt. Bei Überschreiten des Limits fallen entweder zusätzliche Gebühren an, oder die Geschwindigkeit wird stark reduziert.

Sollte Ihre App nicht auf Netzwerkverbindungen verzichten können, überlegen Sie sich, wie die App reagieren soll, wenn keine Internetverbindung mehr besteht.

Checkliste – Was tun bei Verbindungsabbruch?

- ☐ Brauche ich die Verbindung? ?
- ☐ Kann ich temporär auch ohne Internet arbeiten?
- ☐ Fand gerade eine Datenübertragung statt?
- ☐ Wenn ja, ist sie abgeschlossen, oder muss sie wieder neu gestartet werden?
- ☐ Kann ich kommende Anfragen in eine Queue schreiben und beim nächsten Mal gesammelt übertragen?

Wichtig ist in erster Linie, dass die App nicht plötzlich abstürzt oder unerwartet reagiert, sobald keine Internetverbindung mehr besteht.

Standard	Geschwindigkeit	Art
GPRS	55,6 kBit/s	Mobilfunk
Edge	236,8 kBit/s	Mobilfunk

Standard	Geschwindigkeit	Art
UMTS/3G	7,2 bis 28 MBit/s	Mobilfunk
LTE/4G	100 MBit/s	Mobilfunk
WLAN 802.11g	54 MBit/s	Lokales Netzwerk
WLAN 802.11n	600 MBit/s	Lokales Netzwerk
WLAN 802.11ac	>1,3 GBit/s	Lokales Netzwerk
Bluetooth 2.0 + EDR	2,1 MBit/s	Kurzstrecke (ca. 10 m)
Bluetooth 3.0	24 MBit/s	Kurzstrecke (ca. 10 m)
Bluetooth 4.0/Low Energy	24 MBit/s	Kurzstrecke (ca. 10 m)
NFC	424 kBit/s	Kurzstrecke (<10 cm)

Übertragungsgeschwindigkeiten in Funknetzwerken.

UMTS hat eine Abdeckung von über 90 % in Deutschland, bezogen auf die Besiedelung, was aber nur weniger als 50 % der Landesfläche entspricht. Daher muss unterwegs immer damit gerechnet werden, dass die Verbindung plötzlich unterbrochen wird. Das kann passieren, wenn man ein Gebäude oder U-Bahnhof betritt, wenn man sich in einem Funkloch befindet oder weil über denselben Mobilfunkmast zu viele andere Personen gleichzeitig im Netz unterwegs sind.

2.4 Bluetooth

Bluetooth ist eine Kurzstreckenkommunikation zwischen zwei Geräten. Meistens sind es Lautsprecher, Kopfhörer oder Freisprecheinrichtungen, die über Bluetooth kabellos mit einem Smartphone verbunden werden. Ebenso kann man auch mit mehreren Spielern gemeinsam ein Spiel spielen, und die Synchronisation erfolgt über Bluetooth.

Bluetooth 4.0 ist auch die Basis für die von Apple in iOS 7 eingeführten iBeacons. Das ist ein System zur zellenbasierten Ortung innerhalb von Gebäuden. Die platzierten iBeacons senden permanent eine Kennung sowie zwei weitere Werte. Anhand dieser Informationen kann eine App, die für diese iBeacons angepasst ist, bestimmte Informationen anzeigen oder die eigene Position auf einer Karte darstellen. In seinen eigenen Ladengeschäften setzt Apple das System dazu ein, dem Besucher zusätzliche Produktinformationen auf sein Smartphone zu schicken,

Andreas Itzchak Rehberg

Das inoffizielle Android System-Handbuch

Dem Android-System unter die Haube geschaut!
In Zusammenarbeit mit AndroidPIT, der größten deutschsprachigen Community zu Android.

Vorwort

Mein erstes Android-Buch, *Das inoffizielle Android-Handbuch*, erfreut sich nach wie vor großer Beliebtheit, und bereits nach wenigen Monaten war eine Neuauflage notwendig: Der Vorrat im Lager des Verlags war aufgebraucht! Was bleibt mir bei derart großem Interesse anderes übrig, als für eine Fortsetzung zu sorgen?

Was diese zum Thema hat, lässt sich bereits am Namen erkennen: Das Android-System selbst. Es wird insbesondere auf die Konfiguration eingegangen – ebenso aber auch auf Themen wie Backup und Datensicherheit. Wie gewohnt, werden natürlich Werkzeuge (Apps) benannt, mit denen sich das System im Griff behalten lässt, mit denen man es also konfiguriert und ggf. auch repariert. Neben all diesen Apps sollen aber Praxis-Tipps und Hintergründe auch nicht fehlen.

Wo sich all diese (und weitere) Werkzeuge finden lassen? Überwiegend natürlich wieder in meinen *App Reviews nach Einsatzzweck* bei *AndroidPIT*, deshalb wird auch an mancher Stelle für mehr Details oder für weitere Apps zu einem Themenbereich dorthin verwiesen. Im Forum findet sich dann auch wieder ein Thread für Fragen und Anregungen.

http://www.androidpit.de/de/android/forum/thread/409715/

Anders als im *inoffiziellen Android-Handbuch* geht es in diesem Band um mehr als nur einen kurzen Überblick – hier wird tiefer eingestiegen. Dennoch finden sich auch wieder Verweise zu den entsprechenden Themen im Forum, nicht unbedingt zur Vertiefung, aber dort sind die Informationen oft aktueller und vollständiger.

http://www.androidpit.de/

Und noch eins muss ich loswerden: Viele der hier kurz vorgestellten (oder auch nur genannten) Apps habe ich selbst nie getestet – dafür fehlt einfach die Zeit: Würde ich jede App erst selbst testen, wäre ich noch immer mit dem ersten Buch beschäftigt. Und wäre das dann fertig, könnte ich gleich wieder von vorn beginnen – einfach weil die Informationen bis dahin bereits veraltet wären. Ein Grund mehr, auf die Erfahrungen der Community zurückzugreifen: Bei AndroidPIT ist jeder jederzeit herzlich willkommen! Auch für Feedback sowie Fragen zu diesem Buch findet sich dort ein Plätzchen.

Hinweise zur Nutzung des Buches

Wie bereits im Vorwort aufgeführt, handelt es sich bei diesem Buch um eine Fortsetzung meines Buches *Das inoffizielle Android-Handbuch*. Ausgewählte Themen sollen hier vertieft werden. Teilweise setze ich Kenntnisse voraus, die im „ersten Band" behandelt wurden.

Auch wenn die Themen hier ausführlicher behandelt werden, so finden dennoch nicht alle verfügbaren Apps Erwähnung. Das ist auch gar nicht möglich. Nicht nur würde es den Umfang dieses Buches sprengen – es wäre auch schon nicht mehr aktuell, kaum dass es die Buchläden erreicht hätte: Täglich finden hunderte oder gar tausende neue Apps ihren Weg in den Google Playstore. Daher stehen auch die in diesem Buch vorgestellten Apps exemplarisch für die Möglichkeiten, die Android im jeweiligen Zusammenhang bietet. Weitere Apps sind in der Regel im zu Beginn eines Kapitels genannten Forums-Thread zu finden, ebenso wie Kommentare ihrer Nutzer.

Ein weiterer Hinweis gilt den Screenshots: Diese sind oft der Playstore-Seite der jeweiligen App entnommen und daher häufig auf Englisch. Das muss allerdings nicht zwangsläufig heißen, dass selbige App dann nicht auch auf Deutsch läuft – meist ist das der Fall. Es belegt also lediglich, dass sie auch Englisch kann ...

Einige der hier vorgestellten Tools und Apps setzen den root-Zugang (vgl. Abschnitt 2.1) zum Gerät voraus. Stellen, an denen solche Anwendungen beschrieben werden, sind am Seitenrand mit einem Symbol und einem grauen Balken markiert.

Danksagung

Auch diese gehört an den Anfang eines guten Buches. Ich füge sie hier jedoch nicht aus reinem Pflichtbewusstsein ein – sondern weil ich einigen Menschen auch und gerade in Zusammenhang mit diesem Buch wirklich dankbar bin.

Meine liebe Frau möchte ich dieses Mal gleich an erster Stelle nennen: Wiederum fand sie mich über Wochen und Monate hinweg fast nur an der Tastatur vor. Wahrscheinlich sucht sie auch bereits nach einer Möglichkeit, bei ihrem Mann einen „Factory-Reset" durchzuführen oder ein passendes Custom-ROM aufzuspielen (um das Verhalten anzupassen). Danke für Dein großes Herz und Verständnis!

Zu einigen Details hat mich Andrew Hoogs Buch „Android Forensics" inspiriert, das ich dankenswerterweise für den Franzis Verlag übersetzen durfte (es ist mittlerweile unter dem Titel „Android Forensik" auch in deutscher Sprache verfügbar). Andrew und seinem Team, insbesondere auch Kevin, an dieser Stelle meinen besonderen Dank – nicht nur für die Inspiration, sondern auch für den interessanten und fruchtbaren Austausch per Mail zu diversen Details.

Toni, meinem Lektor bei Franzis, sei an dieser Stelle ebenfalls gedankt: Er hat schließlich die Sache mit Andrews Buch eingefädelt. Und mir auch erlaubt, mich für „Das inoffizielle Android-Systemhandbuch“ an den dortigen Themen zu bedienen. Einige kurze Passagen meiner Übersetzung sind daher auch hier eingeflossen.

Weitere dankbare Grüße gelten den Communities bei *AndroidPIT* und *StackExchange* – insbesondere Zuul, t0mm13b, ce4 und Flow. Von Euch habe ich zahlreiche Anregungen erhalten, mit Euch meine Ideen diskutiert (dabei so manche Nacht im Chat verbracht) und viel dabei gelernt. Das sollten wir auf jeden Fall fortsetzen!

Abschließend noch meinen Dank an die Leser dieses Buches, die nun schon bis hierher durchgehalten haben, hoffentlich auch noch ein wenig weiter lesen und aus diesem Buch möglichst großen Nutzen ziehen: Viel Spaß bei der Lektüre – und mit Euren Androiden!

Inhaltsverzeichnis

4 Tuning

4.1 Einleitung und Überblick

Im vierten Teil dieses Buches wenden wir uns schließlich dem Tuning zu: Wie lässt sich möglichst viel aus dem Androiden herausholen?

Zuerst einmal sollte geklärt werden: Möglichst viel wovon? Altbekannt: Wenn ein Manta-Fahrer in den Supermarkt geht, will er Mantarinen und Tunefish. Und später dann die Kiste tiefer legen (ja, auch die auf dem Friedhof). Aber was wollen wir nun mit dem Tunen erreichen?

Prinzipiell gibt es zwei Richtungen: „Schneller, weiter, höher" (Performance/Geschwindigkeit) und „langes Durchhaltevermögen" (Akku-Laufzeit); bis auf einige wenige Ausnahmefälle schließen sich diese Tuning-Ziele i. d. R. gegenseitig aus. Des Weiteren gilt es zu beachten: Was kann „Otto Normalanwender" umsetzen, und was richtet sich eher an „Nerd Root"? Bevor wir uns auf die Details einlassen, zunächst eine Kurzübersicht über die Möglichkeiten:

Generelle Maßnahmen

Toll, dass es Dinge gibt, die beides bedienen: Mehr Speed ohne zusätzliche Akku-Belastung nimmt man gern, längere Akku-Laufzeiten ohne Performance-Einbußen ebenso. Utopisch? Zu wahr, um schön zu sein? Es gibt sie aber wirklich, die derartigen Tuning-Maßnahmen.

Aufräumen:

- nicht mehr verwendete/benötigte Apps deinstallieren
- Cache bereinigen
- selten genutzte Apps am automatischen Starten hindern (ggf. wird root benötigt)
- häufig genutzte Apps möglichst im internen Speicher installieren (auch an die ständig im Hintergrund laufenden Apps denken). Dieser hat einerseits kürzere Zugriffszeiten (ist also schneller) und andererseits weniger Energiebedarf bei Zugriffen.

Live-Wallpaper und sonstige „animierte Dauerrenner": Wer darauf verzichten kann, sollte es tun, denn die nuckeln anständig am Akku und knabbern auch gern an der CPU!

Auf Task-Killer weitgehend verzichten. Diese tragen i. d. R. weder zu verbesserter Performance noch zu längerer Akku-Laufzeit bei – sondern ganz im Gegenteil.

Ein netter Nebeneffekt der ersten beiden Aufräumaktionen: Es wird wieder interner Speicher frei. Eine Nebenwirkung, die man auch gern mitnimmt.

Bessere Akku-Laufzeit

Warum nur ist am Ende des Akkus noch so viel Tag übrig? Eine Frage, die sich sicher manch einer (und nicht nur einmal) schon gestellt hat. Was hier Abhilfe schafft, geht nicht selten zu Lasten der Performance, aber häufig so unmerklich, dass man es durchaus in Erwägung ziehen kann:

- WLAN komplett abschalten, wenn's nicht gebraucht wird. Das spart enorm (siehe Anhang!) und lässt sich auch automatisieren. Die Verzögerung bei der bedarfsmäßigen Aktivierung ist zu verschmerzen.
- Wer kein mobiles High-Speed benötigt: 3G aus, das frisst auch ganz nett. Wer es hin und wieder benötigt, kann es bei Bedarf aktivieren. YouTube per mobilem Datennetz macht mit 2G aber natürlich weniger Spaß.
- GPS benötigt im Standby zwar so gut wie keinen Saft – aber wenn es deaktiviert ist, kann auch keine „böse App" für „Werbestandort" darauf zugreifen.
- Helligkeit des Displays weitmöglichst herunterregeln. „Aufdrehen" dann im Bedarfsfall. Das Display-Timeout möglichst kurz (aber nicht zu kurz) einstellen.
- Die Datensynchronisierung muss evtl. nicht (für alle Apps) ständig laufen (Hintergrunddaten deaktivieren, Intervalle anpassen, Apps ausnehmen).
- Verschiedene Apps helfen ggf. bei der Laufzeit-Verlängerung, indem sie gewisse Aufgaben automatisch ausführen (*JuiceDefender*, *GreenPower*, *BatterySaver* ...).
- Gute Pflege nimmt der Akku gern an (Temperaturbereich, Ladezustand, Kalibrierung).
- *root* und Modder: CPU nicht über-, sondern ggf. eher untertakten. Ich weiß, das klingt nicht besonders „cool" – spart aber Akku. Und auch dies lässt sich automatisieren: So benötigt man beispielsweise bei ausgeschaltetem Display und zu nachtschlafender Stunde meist nicht die volle CPU-Last.

Mehr Speed

Speedy Gonzales überholen mit einem Androiden der ersten Generation? Da stehen die Chancen eher schlecht. Aber neben den ***generellen Tipps*** gibt es auch hier noch die eine oder andere Spezialität:

- RAM bereinigen (Android-interne Einstellungen optimieren). Die persönlich richtige Mischung aus „verfügbarem RAM" und „schnell verfügbaren Apps" festlegen.
- Eventuell Swap-Space nutzen (braucht *root*).
- Nur für bestimmte Situationen, in denen es darauf ankommt: CPU ggf. leicht übertakten. Frisst aber auch mehr Akku. Und benötigt *root*.

4.2 Generelle Tuning-Maßnahmen

Nicht mehr verwendete/benötigte Apps deinstallieren

Ein unnötiger Vorschlag? Verstehe: Auch ein Wildfire oder gar ein Gerät mit noch weniger internem Speicher im Einsatz? Trotzdem nicht für jeden selbstverständlich, aber: Alles raus, was keine Miete zahlt!

Und was soll das bringen? Mehrere Dinge. Zuerst einmal das offensichtliche: Es wird wieder interner Speicher frei, der zuvor von der App belegt wurde, aber auch der, den die Daten der App mit Beschlag belegten, und der Platz, den der Cache der App in Anspruch nahm.

Gerade bei Geräten mit wenig internem Speicher kann das einiges bringen, denn dieser frei gewordene Platz steht nicht nur neuen Apps zur Verfügung, sondern kann auch als Cache genutzt werden (das geschieht automatisch). So muss beispielsweise eine noch aktuelle Datei nicht aus dem Netz neu geladen werden, was schnelleren Zugriff und geringeren Stromverbrauch bedeutet.

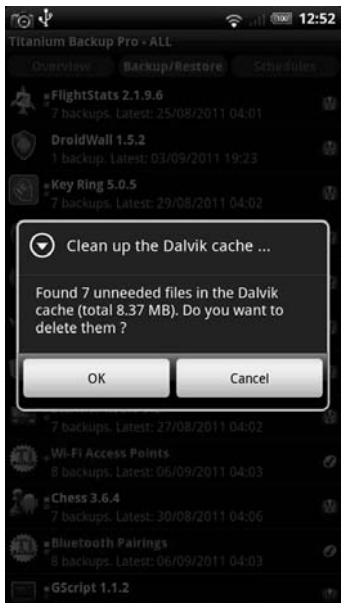


Bild 4.1: ***Titanium Backup*** räumt den Dalvik-Cache auf

Hat man eine Reihe von Apps auf diese Weise ins Nirwana befördert, kann man sich als Wurzelmensch (also Android-User mit *root*) auch Gedanken um die Bereinigung des ***Dalvik-Cache*** machen. Hier sind ja ebenfalls Lücken entstanden, und eine Reorganisati-

on würde für einen schnelleren (und kürzeren, also Akku-schonenderen) Zugriff sorgen. Eine Möglichkeit dazu bietet *Titanium Backup Pro* mit der Option „Cleanup Dalvik Cache" (Bild 4.1). Alternativ bieten die meisten Custom-Recovery-Menüs eine entsprechende Option, den Dalvik-Cache komplett zu löschen – beim nächsten Boot wird er dann automatisch neu aufgebaut.

Ja toll – und wie wird man eine App nun los? Dazu kann man auf die Bordmittel zurückgreifen, von denen üblicherweise zwei zur Verfügung stehen. Auf jedem Androiden findet sich in den Einstellungen die Möglichkeit, die installierten Anwendungen zu verwalten (üblicherweise unter *Einstellungen → Anwendungen → Anwendungen verwalten*). Hier wählt man die zu deinstallierende App aus und betätigt den passenden Button – naheliegenderweise ist dieser meist mit *Deinstallieren* beschriftet. Ist er ausgegraut und lässt sich nicht betätigen, muss man evtl. zunächst die App beenden (über den Button *Stoppen erzwingen*). Klappt es danach noch immer nicht, handelt es sich um eine „System-Applikation" – ohne *root* lässt sich eine solche nicht entfernen.

Das zweite Bordmittel ist die Playstore-App. In dieser sollte sich die zu entfernende App unter *Meine Downloads* wiederfinden und ebenfalls mit einem *Deinstallieren*-Button versehen sein. Ist dies nicht der Fall, wurde die App nicht über den Playstore installiert, und es handelt sich evtl. gar um eine System-App. Da hilft uns die Playstore-App dann nicht weiter.

Als alternative Uninstaller kommen zahlreiche Apps in Betracht. Da wäre zum Beispiel das bereits genannte *AppMonster*, das neben der Sicherung auch eine Deinstallation von Apps ermöglicht. Ebenso kommen spezielle Deinstallations-Apps wie *Fast Uninstaller* oder *Shake-Uninstall* in Frage. Letzteres bietet einen spaßigen Ansatz: Man öffnet die zu entfernende App und schüttelt zum Deinstallieren das Gerät (daher auch der Name). Also einfach immer laufen lassen: Wenn es einen dann bei einer App schüttelt, wird sie automatisch entfernt. Was aber wiederum nur bei selbst installierten Apps klappt. Und Werbung lässt sich so nicht entfernen.

Für die vom Hersteller vorinstallierte sogenannte ***Bloatware*** kann der Wurzelmensch wieder auf das bewährte *Titanium Backup* zurückgreifen. Um dabei nicht zu viel Schaden anzurichten (indem man beispielsweise eine vom System wirklich benötigte App versehentlich löscht) und sich ebenso wenig die Möglichkeit eines ***OTA***-Updates nicht zu verbauen, lassen sich unerwünschte System-Apps damit auch einfach „einfrieren": Sie werden dann nicht gelöscht, sondern lediglich „kalt gestellt" – sind also schlicht „unsichtbar", werden nicht mehr gestartet und stören damit auch niemanden. Eine Möglichkeit, die übrigens auch das kostenlose (und nur ca. 150 kB große) ***App Quarantine*** anbietet.

App Quarantine

Eine abschließende gute Nachricht: Ab *Ice Cream Sandwich* (Android 4.0) ist die Funktionalität zum „Einfrieren" von Bloatware zur Nutzung durch den Endanwender von Haus aus ins System integriert.

Cache bereinigen

So ein Cache ist schon etwas Feines. Er sorgt nicht nur für schnellere Zugriffe, sondern spart dabei auch gleich noch CPU, Traffic und Akku. Was also ist daran so schmutzig, dass es bereinigt werden muss?

Dafür ganz kurz die allgemeine Erklärung, wie ein Cache funktioniert. Es handelt sich dabei um eine Art „temporären Zwischenspeicher", beispielsweise für Dateien, die aus dem Internet geladen werden (wenn lokal vorhanden und aktuell, spart das ein kosten- und zeitintensiveres Nachladen) oder um die Ergebnisse einer aufwendigeren Berechnung (spart dann Zeit, CPU und Akku) zwischenzuspeichern. Soweit ist das alles prima und überaus nützlich.

Problematisch wird es erst dann, wenn kein Verfallsdatum festgelegt wurde und keine automatische Bereinigungsaktion stattfindet – dann sammelt sich im Cache nämlich überwiegend „Hausmüll", den niemand mehr braucht. Der wiederum bremst den Zugriff auf sinnvolle Cache-Daten und belegt wertvollen Speicherplatz, was leider keine Seltenheit ist. Also sprach Zarathustra: Ab und an einmal Hausputz betreiben kann hilfreich sein. Nicht zu oft natürlich, sonst ist der Cache ja insgesamt nutzlos. Wie oft eine Bereinigung sinnvoll ist, lässt sich pauschal schwer sagen – doch spätestens wenn der interne Speicher knapp wird, ist dies sicher eine der ersten zu ergreifenden Maßnahmen. Negative Seiteneffekte stehen nicht zu befürchten: Findet eine App das gewünschte Element nicht im Cache, wird es einfach neu heruntergeladen bzw. generiert, was auch der Grund ist, warum eine Bereinigung manchmal eine zickige App wieder in die Bahn bringt: Kaputte Elemente im Cache können so etwas verursacht haben. Sind die weg, können sie nicht mehr länger für Probleme sorgen.

Wo geht es nun zum Besenschrank? Mit Bordmitteln heißt es: Unter *Einstellungen → Anwendungen → Anwendungen verwalten* alle Apps einzeln anwählen und schauen, ob und wie viel Cache sie belegen sowie ggf. den passenden Button zum Leeren des entsprechenden Cache betätigen. Es gibt keine Übersicht, die vielleicht gar noch nach belegter Datenmenge sortiert wäre.

Bis vor Kurzem war das auch das Ende der Fahnenstange für den „Normal-Anwender". Dinge wie „Alle Caches mit einem Klick löschen" blieben „Nerd Root" vorbehalten. Aber das hat sich glücklicherweise geändert.

So bietet beispielsweise ***Quick App Manager*** (Bild 4.2) mehr als nur die Möglichkeit, alle Caches mit einem Klick zu putzen. Hierfür lässt sich sogar ein Scheduler einrichten, der das automatisch im eingestellten Intervall erledigt und sich nebenbei auch um weitere „historische Daten" (wie den Verlauf des Webbrowsers) kümmert. Wer das nicht ganz so vollautomatisch mag, kann sich auch bei Erreichen eines konfigurierbaren Limits warnen lassen, etwa wenn der Cache mehr als fünf Megabyte belegt, und dann selbst in der sortierten Liste nachschauen

Quick App Manager

und entscheiden, welche App man bereinigen möchte. Nebenbei kann *Quick App Manager* auch Apps mit potentiell risikobehafteten Permissions anzeigen und bringt einen App2SD sowie einen Task-Manager mit. Für diesen Komfort ist allerdings ein knapper Euro fällig – eine gratis Test-Version konnte ich nicht finden.

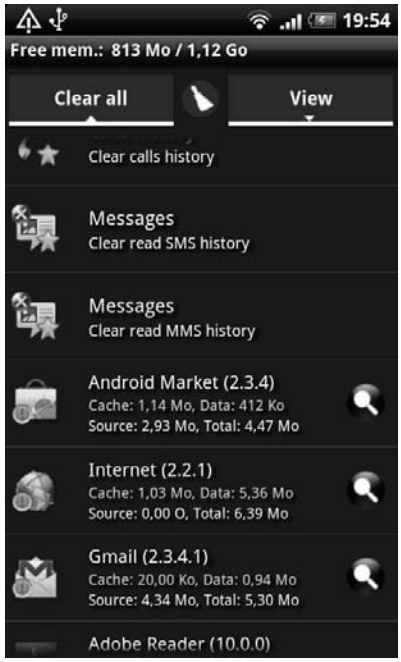


Bild 4.2: ***Quick App Manager*** bereinigt alle Caches mit nur einem Klick

Quick Cache Cleaner

Wer auf die Extra-Manager sowie den Scheduler verzichten kann, findet in ***Quick Cache Cleaner*** eine mögliche Alternative. Um Cache und Browserverlauf kümmert sich diese App ebenfalls mit nur einem Klick, auch die Sortierung (nach Name bzw. belegtem Cache) sowie die Einschränkung auf Apps, die überhaupt Cache belegen, beherrscht sie. Und sie ist darüber hinaus auch noch gratis erhältlich.

1Tap Cleaner

Ebenfalls gratis (oder besser: werbefinanziert) gibt es mit ***1Tap Cleaner*** (Bild 4.3) eine weitere App, die sich sowohl um den Cache als auch um die anderen „historischen Fälle“ kümmert und dabei auch einen Scheduler mit an Bord hat. Nebenbei kümmert sie sich auch um die sogenannten „App Defaults“ und setzt sie bei Bedarf zurück. Das ist hilfreich, wenn man beispielsweise einen alternativen Launcher zum Standard gemacht hat und nicht weiß, wie man diesen Standard nun wieder los wird, um den „alten“ Launcher nutzen zu können (klar geht das auch in dessen Einstellungen unter *Anwendungen verwalten*

– aber warum lange suchen?). Neben diversen Sortiermöglichkeiten lässt sich bei dieser App die Liste der Anwendungen sogar nach Namensbestandteilen filtern. Wer die Gratis-Version ausgiebig getestet hat und den Entwickler belohnen (oder einfach nur die Werbung loswerden) möchte: Mit einem Euro ist man dabei.

Bild 4.3: Auch ***1Tap Cleaner*** bereinigt die Caches per Knopfdruck

Selten genutzte Apps am automatischen Starten hindern

Ein altbekanntes Lied, nicht nur auf unseren geliebten Androiden: So manche App fühlt sich so furchtbar wichtig, dass sie meint, ständig laufen zu müssen, am besten bereits vor dem Booten. Am schlimmsten sind dabei die Ach-so-tollen Apps, mit denen uns Hersteller und Provider „zwangsbeglücken" und die „Otto Normalanwender" nicht einmal beseitigen (sprich: deinstallieren) kann. Kaum ist der Androide gestartet, schon laufen auch Flickr, FM-Radio, Google Maps, Peep ..., ohne dass wir sie darum gebeten hätten. Sie belegen dabei Speicher und verbraten CPU und Akku gleichermaßen.

Da möchte man gern einmal kräftig dazwischenschlagen. Wenn man diese Bösewichte nicht einfach deinstallieren kann (etwa weil es sich um System-Apps handelt und man keine root-Rechte hat oder weil man die betreffende App doch hin und wieder einmal benötigt), kann man diesen Wahnsinn nicht einfach abstellen? Bei mancher App findet sich ein betreffender Punkt in den Einstellungen. Ist dies jedoch nicht der Fall, sieht es ohne *root* recht mau aus.

Ein einfacher Task-Manager ist hier definitiv der falsche Ansatz. Allerdings gibt es einige Apps, mit denen sich das automatische Starten unterbinden lassen soll. Ohne root-Rechte sieht das aber eher so aus, dass die betroffenen Apps nach dem Auto-Start automatisch gekillt werden – und das führt leider häufig dazu, dass so manche App dann einfach wieder neu startet, was in einem Teufelskreis mit Akkufresser und CPU-Heizer enden kann. Es sei denn, man hat sich für ***Autorun Manager*** (Bild 4.4) entschieden: Diese App kann derartige Kreisläufe erkennen und sieht sodann von einer weiteren Behandlung ab. Als Anwender sollte man dann ebenfalls davon Abstand nehmen, eine Behandlung erzwingen zu wollen.

Autorun Manager

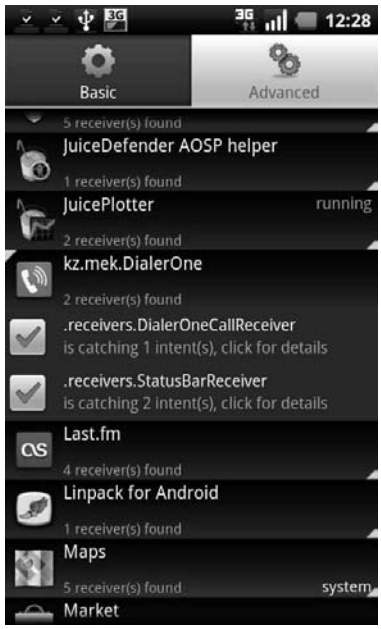


Bild 4.4: Im root-Modus kann ***Autorun Manager*** alle Startrampen gezielt ansprechen

Weitaus bessere Chancen hat man, wenn man mit root-Rechten ausgestattet ist – und zwar beispielsweise mit gerade genannter App. Diese ist im erweiterten („Advanced“) Modus nämlich in der Lage, automatische App-Starts von vornherein zu unterbinden. Um automatisch gestartet zu werden, muss eine App sich nämlich bei ihrer Installation für die gewünschten Ereignisse (sogenannte „Intents“) registrieren. Nur mit vollen Systemrechten (oder per Deinstallation der App) kann man diese Registrierung wieder auflösen. Genau das geschieht an dieser Stelle, und so hat man volle Kontrolle darüber, wann eine App automatisch starten darf. Nicht nur das „Nach-dem-Booten-Starten“ lässt sich hier abschalten, sondern beispielsweise auch Dinge wie „bei eingehendem Anruf“ oder „nach hergestellter WLAN-Verbindung“, „USB angeschlossen“ ...

Natürlich lassen sich einzelne „Intents“ unabhängig voneinander deaktivieren und diese Aktion später ggf. wieder rückgängig machen, sollte dies notwendig werden. Zu beachten ist jedoch: Vor Deinstallation des *Autorun Manager* sollte man die ursprünglichen Einstellungen wieder herstellen – andernfalls ist dies nur durch Neuinstallation der betroffenen Apps möglich, was sich bei System-Apps als zumindest schwierig darstellen dürfte.

Autorun Manager ist aus meiner Sicht in diesem Bereich die beste Wahl, zumal es auch noch gratis verfügbar ist. Als Alternative käme eventuell noch ***Autostarts*** (Bild 4.5) in Frage, das nach dem gleichen Prinzip arbeitet (Deaktivieren von Intents), jedoch knapp einen Euro kostet. Auch bei dieser App gilt es, vor der Deinstallation die ursprünglichen Einstellungen wiederherzustellen.

Autostarts

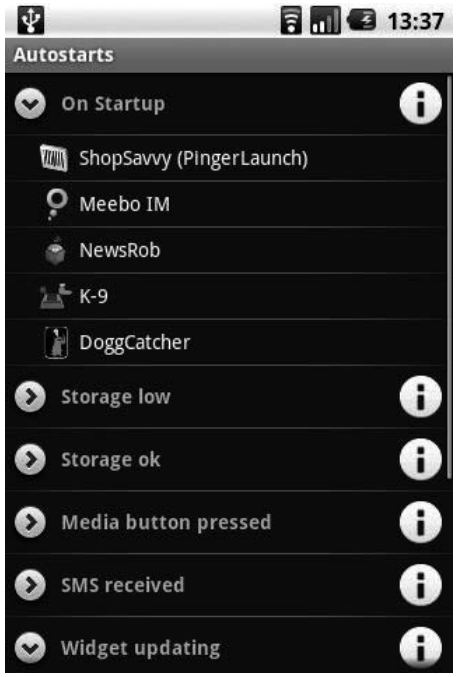


Bild 4.5: Auch ***AutoStarts*** kann unerwünschte Starts verhindern

Jetzt sollte man aber nicht einfach wild alles abschalten, was einem vor die Flinte (oder besser: unter die Finger) kommt – von System-Apps am besten ganz die Finger lassen (solange man nicht weiß, was man da tut). Sollte eine App sich anschließend irgendwie seltsam verhalten, kann man natürlich alles wieder rückgängig machen (oder die betroffene App einfach neu installieren). Aber warum gleich von vornherein für Probleme sorgen? Wenn man sich auf Dinge beschränkt, die relativ klar und einleuchtend sind, lassen sich Probleme vermeiden. Besonders Ausschau halten sollte man nach folgenden Intents:

- *After Startup* (Nach dem Booten), aka *BOOT_COMPLETED*: Hier soll etwas nach dem Booten (also nach jedem Systemstart) geschehen.
- *Connectivity Changed* (Netzwerkstatus geändert): Darauf lauscht u. a. der Location-Service von *Google Maps*, um die neuen Zellen zur Übertragung an Google im *LocationCache* abzulegen.

Natürlich sollte man sich hier jeweils fragen, ob das nicht eventuell doch sinnvoll sein kann. In der Regel hat man jedoch bereits einen „Dauerbrenner auf dem Kieker", der ständig läuft und nuckelt – womit sich die Frage meist erübrigt.

Wer sich einfach nur einen Überblick verschaffen möchte, welche App sich für welches Ereignis im System registriert hat, kann ebenfalls das gerade genannte *Autostarts* verwenden: Auch ohne root-Rechte lässt sich diese App installieren, arbeitet dann aber nur im reinen Lese-Modus.

Um auch den „Otto Normalbenutzern" ohne *root* noch eine Alternative zu nennen: Da wäre ***Startup Auditor***, der Apps nach ihrem Auto-Start automatisch wieder abschießt – auf Wunsch auch mehrfach. Um eventuelle „Endlos-Schleifen" (Start-Kill-Start-Kill-Start ...) muss der Anwender sich bei dieser App jedoch selbst kümmern.

Startup Auditor

Bild 4.6: ***Startup Auditor*** beendet Apps, die nach dem Booten unerwünscht starten, sofort nach ihrem Autostart wieder

Und bitte nicht wundern, wenn die Nicht-root-Apps scheinbar weniger automatisch startende Apps finden, als *Autostarts* im Lesemodus anzeigt: Das liegt daran, dass erstere nicht alle „Startrampen" prüfen, sondern sich auf die gängigsten beschränken. Die volle Power bleibt also wieder einmal „Nerd Root" vorbehalten.

Christoph Prevezanos

Das Android-Praxisbuch

für Smartphone und Tablet

Individuelle Anpassungen:
Homescreen, Lockscreen, Komplettsanierung
Datenverwaltung und Synchronisation:
Thunderbird, Outlook, Apple iCal und mehr
Alle Apps sicher im Griff:
E-Mail, Multimedia, Facebook & Co.

Inhaltsverzeichnis

4 Behalten Sie Ihre Apps im Griff

Android erlaubt das Installieren und Verwalten eigener Programme, ganz ähnlich wie auf Ihrem Heimcomputer. Bei Smartphones hat sich für »Programm« der englische Begriff »Application« eingebürgert ... und davon sogar nur die Kurzform »App«. Man installiert also kein Programm, sondern eine App. Die wichtigste Quelle für neue Apps stellt der offizielle Android-Onlineshop von Google dar. Bis März 2012 wurde noch vom »Android Market« gesprochen, jetzt heißt er offiziell »Google Play Store«. Er integriert nun auch die anderen Google-Angebote wie Musik, E-Books und einiges mehr. Unabhängig vom Namen ist aber immer dasselbe gemeint – der Android-Onlineshop.

4.1 Den Google Play Store bequem per PC nutzen

Den Google Play Store können Sie direkt von Ihrem Smartphone aus nutzen. Sie finden ihn mit dem Symbol *Play Store* im Anwendungsfenster. Die Handhabung ist weitestgehend selbsterklärend und einfach. Tippen Sie auf der Startseite eine Empfehlung an, blättern Sie nach rechts und links durch die Kategorien oder suchen Sie mit dem Lupensymbol nach einer App. Das ist gut und unterwegs auch ganz praktisch. Möchten Sie den Play Store hingegen einfach mal in aller Ruhe durchstöbern, ist das recht unhandlich. Viel besser ist es, wenn Sie den Google Play Store am PC nutzen.

Den Google Play Store als App verwenden.

Sie können den Google Play Store am PC allerdings nur nutzen, wenn Ihr Smartphone mit Ihrem Google-Konto und mit dem Store verknüpft wurde. Damit das passiert, müssen folgende Voraussetzungen erfüllt sein beziehungsweise Schritte durchgeführt werden.

- Auf Ihrem Smartphone muss Ihr Google-Konto bei der Einrichtung hinterlegt worden sein. Sie sehen also Ihr Adressbuch, den Kalender, Ihr Google Mail usw. Das sollte eigentlich immer der Fall sein.
- Zusätzlich müssen Sie auf Ihrem Smartphone einmal den Play Store geöffnet und eine App heruntergeladen haben. Das kann irgendeine kostenlose App oder eine Demo sein. Die Hauptsache ist, dass Sie das Bestell- und Installationsprozedere einmal durchlaufen haben.

1. Starten Sie auf Ihrem Desktop-PC oder Notebook Ihren Webbrowser und öffnen Sie den Onlineshop. Die Adresse lautet *https://play.google.com/store*.
2. Oben rechts müssen Sie sich mit Ihrem Google-Konto anmelden. Besitzen Sie mehrere Konten, müssen Sie dasselbe Konto verwenden wie auch auf dem Smartphone.
3. Jetzt können Sie im Play Store stöbern. Verwenden Sie im linken Bereich die *Top Charts*, die *Kategorien*, die *Empfehlungen* usw. Der Play Store funktioniert ganz ähnlich wie andere Onlineshops auch.

Der Google Play Store im Webbrowser.

4. Um eine App genauer anzuschauen, klicken Sie deren Symbol oder Namen an. Jetzt erhalten Sie detaillierte Informationen zu den Funktionen, können Berichte anderer Nutzer lesen und einiges mehr.

5. Auf der linken Seite wird Ihnen angezeigt, ob diese App mit Ihrem Smartphone kompatibel ist. Um diese App nun auf Ihr Gerät zu laden, klicken Sie auf die Schaltfläche *Installieren*.

6. Dadurch öffnet sich ein neues Fenster. Es zeigt Ihnen das verknüpfte Google-Konto, Ihr Smartphone und die wichtigsten Details der App noch einmal an. Klicken Sie unten rechts auf *Installieren*, und schon ist die App bestellt.

Der Google Play Store im Webbrowser.

7 Jetzt wird diese App automatisch an Ihr Smartphone geschickt. Achten Sie also darauf, dass Sie eine Datenverbindung besitzen – entweder per Mobilfunk oder per WLAN.

8 Es dauert nur wenige Sekunden, und schon sehen Sie oben in der Statusleiste das Symbol einer Einkaufstasche und einen Hinweis auf die neue App.

Sehen Sie in das Anwendungsfenster Ihres Smartphones. Dort ist die neue App jetzt mit ihrem Symbol aufgelistet und kann sofort verwendet werden. Tippen Sie das Symbol an, um die App zu starten.

Die neue App wird installiert und ist nun verfügbar.

Weitere Onlineshops auf Ihrem Smartphone
Offiziell gibt es nur einen Onlineshop für Android-Apps, und das ist der »Google Play Store« – früher Android Market. Trotzdem werden Sie auf Ihrem Smartphone vielleicht noch andere vorinstallierte Shopsysteme finden. Alle großen Gerätehersteller wollen am App-Boom mitverdienen und statten ihre Geräte mit zusätzlichen, eigenen Shops aus. Dann kaufen Sie Ihre Apps also bei Samsung, Sony, Motorola, HTC usw.
Dafür benötigen Sie ein weiteres Kundenkonto, denn mit dem Google-Dienst hat das nichts zu tun. Deshalb werden diese Apps auch nicht im Webbrowser oder in der Play Store-App aufgelistet. Ob Sie diesen Herstellershop nutzen möchten, bleibt Ihnen überlassen – notwendig ist es nicht. Manche Hersteller bieten ein paar kostenlose Willkommens-Apps oder Spiele an, die sonst bei Google ein paar Euro kosten würden. Aber ob das eine zusätzliche Registrierung rechtfertigt?

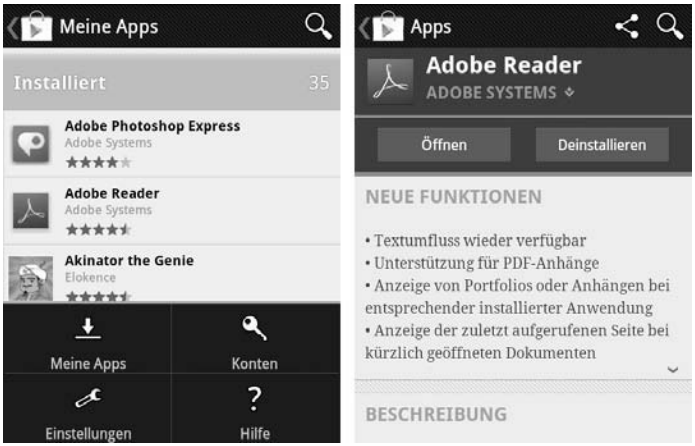
4.2 Ihre Apps und Bestellungen im Store verwalten

Im Laufe der Zeit werden Sie sicherlich eine ganze Menge Apps und Widgets aus dem Play Store herunterladen. Damit Sie dabei nicht den Überblick verlieren, bietet Google eine übersichtliche Liste mit allen von Ihnen heruntergeladenen und installierten Apps.

1. Auf Ihrem Smartphone öffnen Sie die Play Store-App und betätigen einmal die Taste *Menü*. Wählen Sie den Punkt *Meine Apps* aus.
2. Jetzt erhalten Sie eine Liste mit allen Apps. Die Liste ist alphabetisch sortiert, und Sie können sich mit dem Finger darin nach oben und unten bewegen.
3. Tippen Sie auf den Namen einer App, können Sie sich die detaillierten Informationen anzeigen lassen sowie die Bewertungen anderer Anwender.

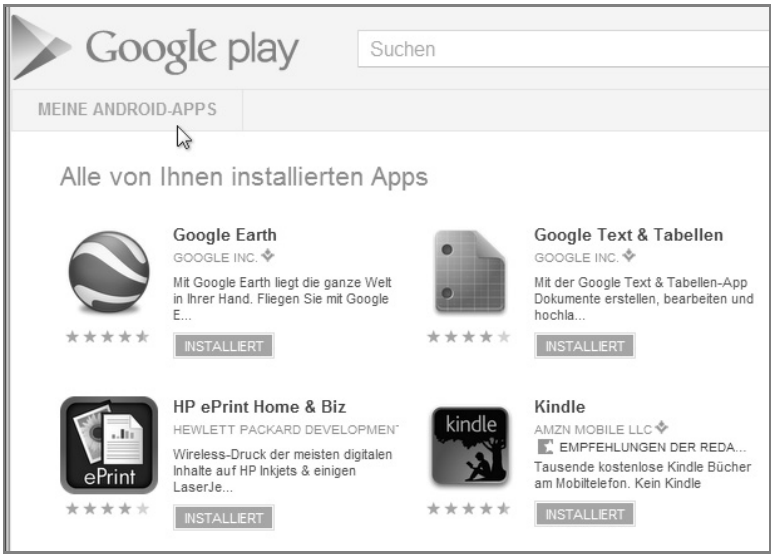
Auch hier ist die Bedienung am Smartphone zwar ganz okay, sodass Sie unterwegs schnell mal etwas bei Ihren Apps nachschauen können. So richtig praktisch und übersichtlich ist das aber nicht. Deshalb sollten Sie erneut Ihren Webbrowser am PC nutzen und Ihre Apps bequem am großen Bildschirm verwalten.

1. Starten Sie Ihren Webbrowser und gehen Sie wieder auf die Startseite des Google Play Store – *https://play.google.com/store*.



Installierte Apps im Smartphone anzeigen.

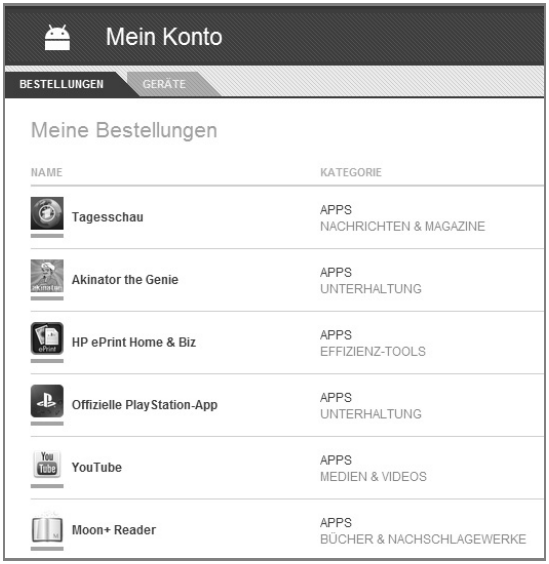
2 Klicken Sie oben rechts auf die Schaltfläche *MEINE ANDROID-APPS*. Sie gelangen auf eine Übersichtsseite, die alle Apps auflistet, die derzeit auf Ihrem Smartphone installiert sind.



Ihre installierten Apps im Webbrowser betrachten.

3. Der Google Play Store listet Ihnen nicht nur auf, welche Apps derzeit auf Ihrem Smartphone installiert sind. Er merkt sich auch, welche Apps Sie jemals über das Smartphone oder den Webbrowser heruntergeladen haben.

4. Klicken Sie hierzu ganz unten auf der Seite auf den Link *Meine Bestellungen und Geräte*. Die angezeigte Liste kann unter Umständen sehr lang sein. Google vergisst nie etwas, und so findet sich hier wirklich alles, was jemals im Google Play Store an Aktivität stattgefunden hat.



Alle jemals heruntergeladenen Apps.

Der Google Play Store ist groß und kompliziert, er stellt selbst für einen Branchenriesen wie Google eine Herausforderung dar. Deshalb gibt es noch ein paar Probleme und Ungereimtheiten, die bis heute nicht gelöst wurden. Wundern Sie sich also nicht, wenn bei Ihnen diese oder ähnliche Probleme auftreten.

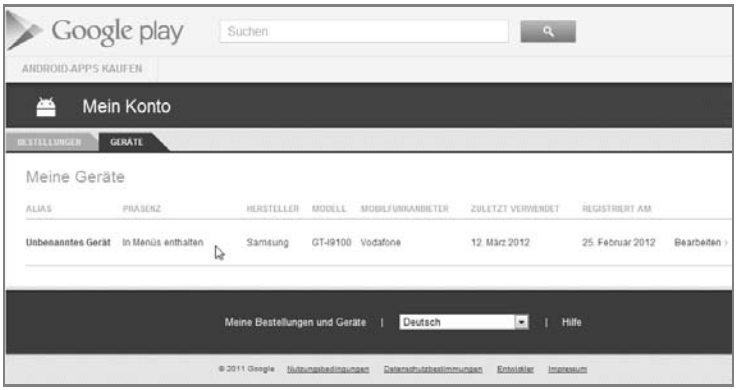
- Nicht alle Apps entfernen sich sauber aus der Liste der installierten Apps. Es kann durchaus vorkommen, dass dort ein Programm erscheint, das längst gelöscht wurde.
- Ebenso tauchen in der Liste einige Apps auf, die Sie niemals installiert haben. Meist handelt es sich um Anwendungen, die der Hersteller Ihres Smartphones vorinstalliert hat. Google erkennt das und listet diese Apps auf.

- Wird eine deinstallierte App fälschlicherweise weiterhin angezeigt, haben Sie derzeit keine Möglichkeit, diese aus der Liste zu entfernen.
- Genauso sieht es mit der Liste der Bestellungen aus. Diese Liste lässt sich nicht bearbeiten, und einzelne App-Einträge können nicht gelöscht werden. Unangenehme oder anstößige Apps bleiben auch in Zukunft in Ihrem Konto dokumentiert.

4.3 Ihre Geräte im Google Play Store verwalten

Sobald Sie Ihr Smartphone das erste Mal im Google Play Store verwenden, wird es automatisch mit Ihrem Google-Konto verknüpft. Auf diese Weise arbeiten Ihr Google-Konto, der Onlinestore sowie alle anderen Google-Dienste optimal mit Ihrem Smartphone zusammen. Sie haben auch die Möglichkeit, mehrere Geräte mit Ihrem Google-Konto zu verknüpfen und zu verwalten. Das können z. B. mehrere Smartphones sein, ein zusätzlicher Tablet-PC oder einfach ein neues Ersatzgerät. Innerhalb des Google Play Store finden Sie extra eine Verwaltungsfunktion für Ihre Geräte.

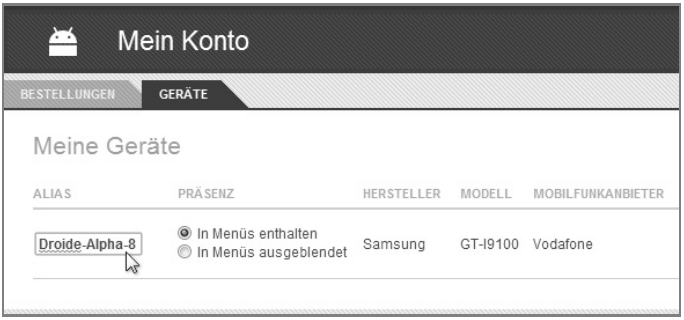
1. Gehen Sie mit Ihrem Webbrowser in den Google Play Store und klicken Sie ganz unten auf den Link *Meine Bestellungen und Geräte*.
2. Sie gelangen auf eine Verwaltungsseite. Klicken Sie dort auf das Register *Geräte*, um alle mit diesem Konto verknüpften Geräte anzuzeigen.
3. In dieser Liste sehen Sie die wichtigsten Daten zu Ihrem Smartphone oder Ihrem Tablet, z. B. den Hersteller, das Modell, Ihren Mobilfunkanbieter, das Registrierungsdatum im Store usw.



Ihre Geräte im Google Play Store.

4. Sie können Ihren Geräten Namen geben, um sie besser unterscheiden zu können – technisch hat das keinerlei Funktion. Klicken Sie dafür ganz rechts auf den Link *Bearbeiten* und tippen Sie ganz links unter *ALIAS* eine Bezeichnung ein.

5. Möchten Sie eines Ihrer Geräte nicht im Play Store anzeigen lassen und Apps damit installieren, können Sie unter *PRÄSENZ* den Punkt *In Menüs ausgeblendet* auswählen. Ihr Gerät bleibt dann auf Sie registriert, erscheint aber nicht im Onlineshop.



Ihre Geräte im Google Play Store.

6. Gehen Sie anschließend wieder in den Google Play Store und bestellen eine App, verändert sich das Fenster ein wenig. Sie sehen dort jetzt den Gerätenamen, den Sie zuvor für Ihr Smartphone vergeben haben.

7. Besitzen Sie mehrere Android-Geräte, können Sie in einer Liste auswählen, an welches Gerät diese App geschickt werden soll. Dabei ist der selbst festgelegte Name sehr hilfreich.

8. Nachdem Sie auf *Installieren* geklickt haben, wird die App wie gewohnt an Ihr Gerät geschickt und dort installiert.

Das gewünschte Gerät mit Namen auswählen.

Googles Regeln für die Nutzung von Android-Geräten

Geräteanzahl – Sie können beliebig viele Geräte mit Ihrem Google-Konto verknüpfen und gleichzeitig nutzen.

Typen – Es können auch verschiedene Gerätetypen gemischt werden, wie z. B. Smartphones und Tablet-PCs.

Apps – Die heruntergeladenen und gekauften Apps dürfen Sie mit Ihrem Konto auf allen Geräten nutzen, soweit diese kompatibel sind – z. B. bei Tablet-PCs.

Ersatzgeräte – Ein Ersatzgerät wird wie ein Neugerät behandelt. Registrieren Sie es und laden Sie alle Apps neu herunter.

Altgeräte – Derzeit können nicht mehr verwendete Geräte nicht aus dem Google-Konto gelöscht werden. Sie bleiben dauerhaft verknüpft. Sie können ein Altgerät nur in den Einstellungen unsichtbar schalten, es aber nicht ganz entfernen.

Christoph Prevezanos

Das Android Tablet-Buch

Systemoberfläche tunen
Optimaler Datenfluss
Fotos bearbeiten

Inhaltsverzeichnis

2 Den Homescreen anpassen

Die Oberfläche von Android wird als „Homescreen“ bezeichnet und ist mit dem Desktop am PC vergleichbar. Über den Homescreen bedienen Sie nicht nur Ihr Tablet, dort befinden sich auch wichtige Informationen, Mini-Programme (Widgets), Verknüpfungen u. v. m. Insgesamt besteht der Homescreen aus fünf Seiten. Vom zentralen Homescreen aus blättern Sie einfach nach rechts und links, um zu den anderen zu gelangen. Dieses Kapitel zeigt Ihnen, wie Sie den Homescreen individuell an Ihre Wünsche anpassen.

2.1 Gestalten Sie Ihren eigenen Hintergrund

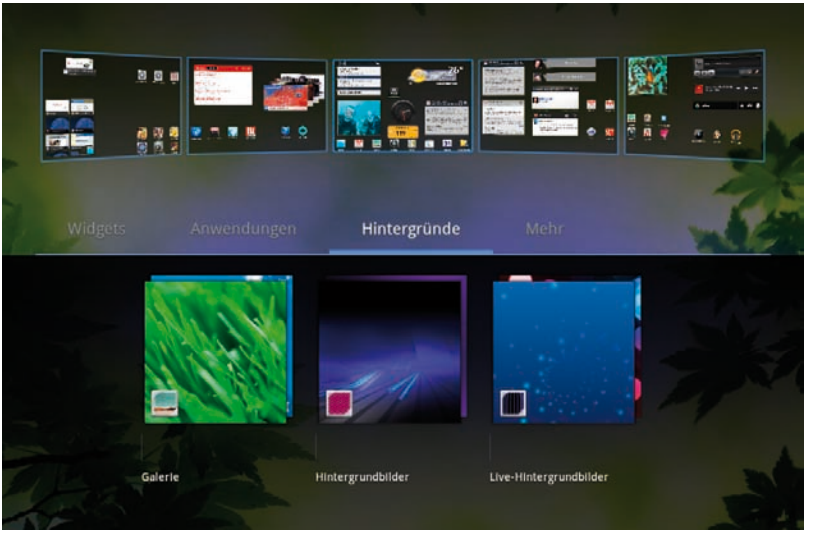
Damit Ihr Tablet genauso aussieht, wie Sie es gern hätten, sollten Sie es als Erstes mit einem individuellen Hintergrundbild ausstatten. Das geht schnell und völlig unkompliziert. So schmücken Sie Ihr Gerät mit Ihrem Lieblingsmotiv, und jeder sieht sofort, wem dieses Tablet gehört. Das gewählte Hintergrundbild gilt automatisch für alle fünf Homescreens Ihres Tablets. Dabei bewegt sich das Bild beim Blättern automatisch ein wenig mit und erzeugt so eine gewisse Räumlichkeit. Android bietet Ihnen dabei gleich drei verschiedene Arten von Hintergrundbild an.

1. Auf dem Homescreen finden Sie oben rechts eine Schaltfläche mit einem Pluszeichen. Diese öffnet die Konfiguration für den Homescreen. Tippen Sie sie einmal an.

Die Konfiguration des Homescreens wählen.

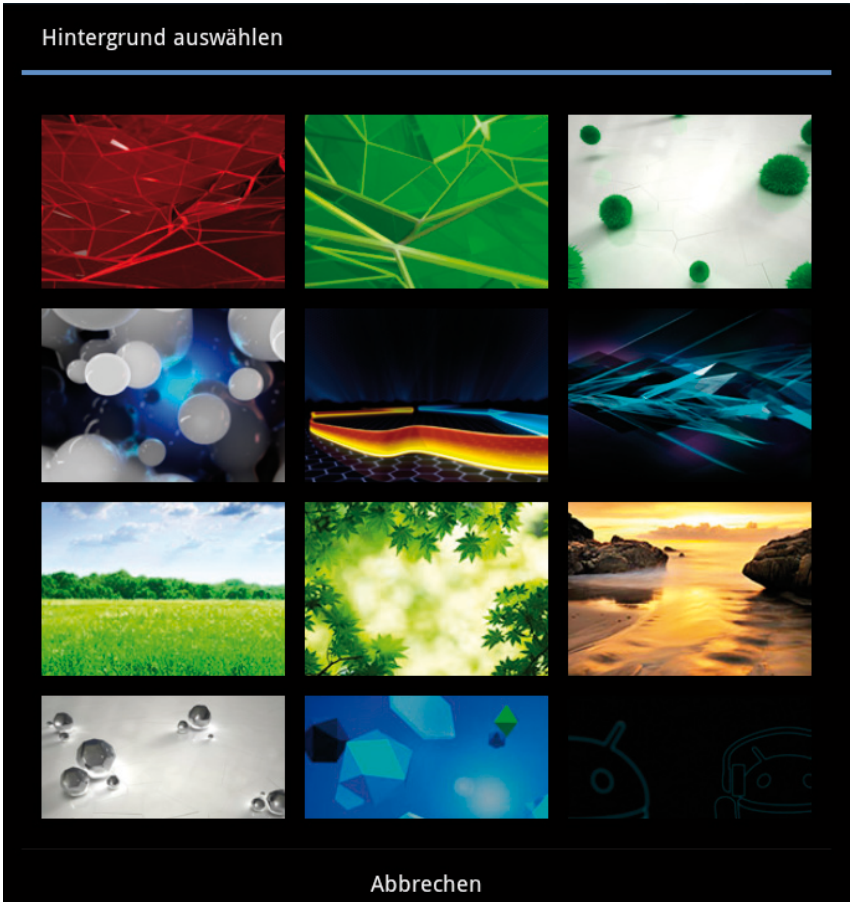
❷ Alternativ können Sie auch auf eine freie Stelle des Homescreens tippen und die Finger einen Moment festhalten.

❸ In jedem Fall gelangen Sie jetzt in die Konfiguration des Homescreens. Tippen Sie in der unteren Zeile das Register *Hintergründe* an.



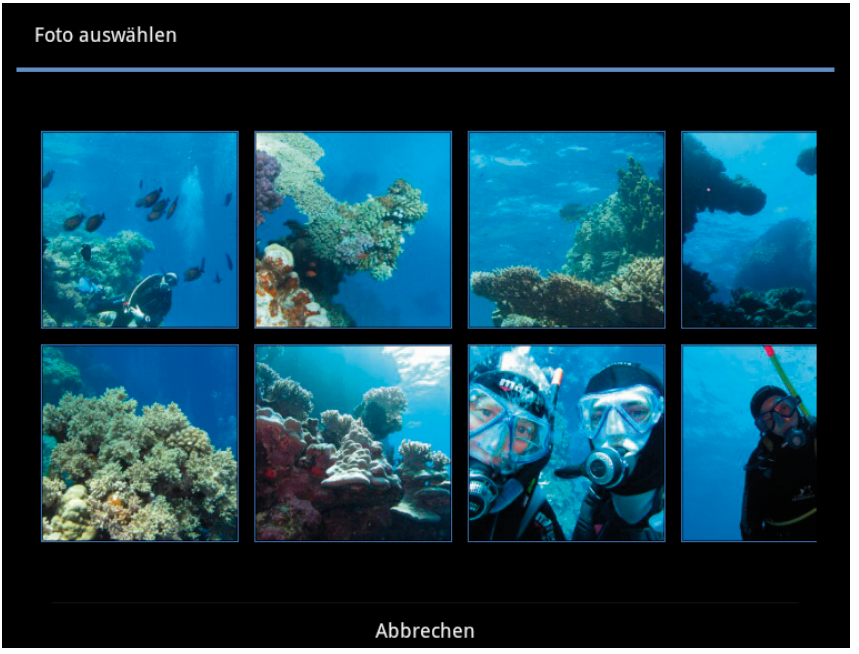
Die Art des Hintergrunds auswählen.

4 Android bringt bereits ein paar Hintergrundbilder mit. Wählen Sie hierfür die Option *Hintergrundbilder* aus. Dadurch gelangen in eine Liste mit vorinstallierten Bildern, die Sie nach oben und unten durchscrollen können. Tippen Sie das gewünschte an, und schon erscheint es als neues Hintergrundbild.



Wählen Sie ein Hintergrundbild aus.

5 Möchten Sie lieber eines Ihrer eigenen Fotos verwenden, tippen Sie auf die Option *Galerie*. Dadurch gelangen Sie in eine Galerieansicht mit allen auf dem Tablet gespeicherten Fotos. Tippen Sie zunächst auf einen Ordner und wählen Sie dort das gewünschte Foto durch Antippen aus.



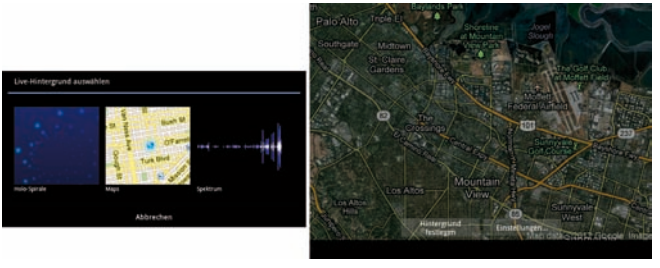
Wählen Sie ein eigenes Foto aus.

6 Das Foto wird nun innerhalb eines Rahmens angezeigt, mit dem Sie den gewünschten Ausschnitt für den Hintergrund wählen. Dabei besteht der Rahmen aus einem Kreuz, immerhin muss das Bild auch beim Kippen des Tablets noch passen. Mit den vier Schiebern an den Außenkanten passen Sie die Bildgröße Ihren Wünschen entsprechend an. Zuletzt tippen Sie oben rechts auf die Schaltfläche *Speichern*.

Legen Sie den Bildausschnitt fest.

7 Android gibt Ihnen auch die Möglichkeit, einen animierten Hintergrund zu wählen. Tippen Sie hierzu auf die Option *Live-Hintergrundbilder*. Zunächst stehen hier nur wenige Animationen zur Auswahl, aber Sie können sich später weitere über den Play Store laden. Tippen Sie den gewünschten Live-Hintergrund an, um eine Vorschau zu sehen.

8 Einige Live-Hintergründe lassen sich über die Schaltfläche *Einstellungen* anpassen. Das kann z. B. die Art der Animation sein, die Farbe, Ihr aktueller Standort oder Ähnliches. Zuletzt klicken Sie auf die Schaltfläche *Hintergrund festlegen*, um den Live-Hintergrund zu aktivieren.



Live-Hintergründe auswählen und anpassen.

2.2 Widgets: Miniprogramme auf dem Homescreen

Die Homescreens stellen Ihren Arbeitsbereich auf einem Android-Tablet dar. Dort finden Sie alle wichtigen Programme, Informationen, Links und vieles mehr. Die Gestaltung des Homescreens wird über sogenannte Widgets vorgenommen. Dabei handelt es sich um kleine Programme, die direkt auf dem Homescreen laufen und somit ohne Umwege bestimmte Informationen oder Funktionen bereitstellen.

- Oftmals ist das Widget selbst bereits das Hauptprogramm, z. B. bei einer Uhr, dem Wetterbericht oder einem Nachrichtenticker.
- Häufig greift das Widget aber auch auf die Daten eines großen Programms zu, z. B. auf den Kalender oder Ihr Postfach.
- Viele größere Programme bringen bereits ein oder mehrere Widgets mit, um die Daten auf dem Homescreen einzublenden, z. B. Nachrichten-Apps, Musikplayer etc.

Passen Sie Ihre Homescreens individuell an.

Auf jedem Android-Tablet sind bereits einige Widgets vorinstalliert, die Sie sofort verwenden können. Anzahl und Art hängen vom jeweiligen Gerätehersteller ab. Im Google Play Store finden Sie jede Menge weitere Widgets, die Sie installieren und verwenden können. So sind Ihnen kaum Grenzen bei der Gestaltung Ihres Homescreens gesetzt.

1. Um Ihrem Homescreen ein neues Widget hinzuzufügen, müssen Sie zunächst auf die gewünschte Bildschirmseite wechseln.

2. Jetzt tippen Sie oben rechts auf die Plusschaltfläche und gelangen in die Homescreen-Konfiguration. Tippen Sie im unteren Bereich auf das Register *Widgets*.

Die Verwaltung für die Widgets öffnen.

3. Die Widgets sind in einer Querleiste angeordnet. Bewegen Sie sich mit einer Wischgeste nach rechts und links, um durch die Widgets zu blättern. Um ein Widget auf den zuvor gewählten Homescreen zu setzen, müssen Sie es nur einmal antippen.

4. Möchten Sie das Widget auf einen beliebigen Homescreen platzieren, müssen Sie es mit dem Finger antippen und festhalten. Nach einem kurzen Moment wird das Symbol beweglich und lässt sich oben auf den gewünschten Screen ziehen.

Haben Sie auf Ihrem Homescreen verschiedene Widgets angeordnet, lassen sich diese natürlich auch verändern, verschieben oder entfernen. Auf diese Weise passen Sie Ihren Screen jederzeit neu an, laden zusätzliche Widgets aus dem Store herunter

oder löschen nicht mehr benötigte Anzeigen. Das geht sehr schnell, und dazu sind nur wenige Fingerbewegungen notwendig.

- Wechseln Sie wie gewohnt auf den Homescreen, den Sie umgestalten möchten.
- Jetzt tippen Sie auf das zu bearbeitende Widget und halten es mit dem Finger einen Moment fest. Damit wird das Widget beweglich und kann auf dem Homescreen verschoben werden.
- Schieben Sie das Widget innerhalb des Homescreens an eine beliebige neue Stelle.
- Der Umriss des Widgets sowie das Raster helfen bei der Positionierung.
- Um das Widget auf eine neue Seite zu setzen, schieben Sie es ganz rechts oder links an den Bildrand. Nach einem kurzen Moment springt das Bild auf den benachbarten Screen.
- Befindet sich das Widget an der gewünschten Stelle, lassen Sie es einfach mit dem Finger los, um es abzulegen.
- Ist eine Seite voll oder das Widget sehr groß, kann es unter Umständen nicht verschoben werden und springt automatisch an seine ursprüngliche Position zurück.

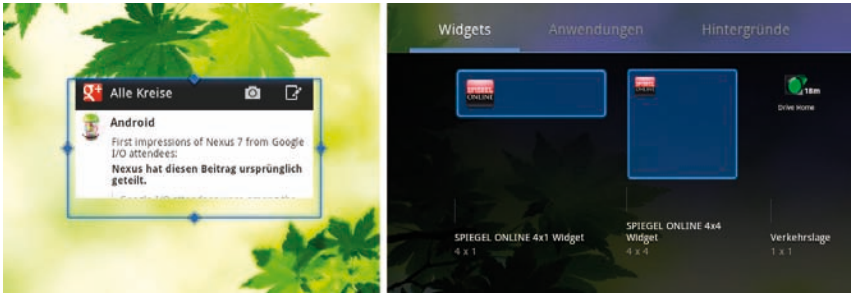
Das Widget auf dem Homescreen verschieben.

5 Möchten Sie ein Widget vom Bildschirm entfernen, schieben Sie es oben rechts auf das Mülleimersymbol für *Entfernen*. Es wird dann nicht mehr angezeigt, bleibt aber auf Ihrem Tablet installiert.

Das Widget vom Homescreen entfernen.

6 Bei vielen Widgets können Sie die Anzeigegröße verändern. Das ist besonders bei Newstickern, dem Posteingang, Fotos etc. sehr praktisch. Tippen Sie dazu das gewünschte Widget an, halten Sie es einen Moment fest und lassen Sie es wieder los. Nun zeigt das Widget einen blauen Rahmen mit Anfasserpunkten. Mithilfe dieser Punkte lassen sich nun Größe und Breite individuell anpassen.

7 Leider unterstützen meist nur speziell für Tablets entwickelte Widgets diese Größenveränderung. Herkömmliche Widgets für Smartphones können das in der Regel nicht. Stattdessen bieten aber viele Widgets beim Hinzufügen direkt verschiedene Größen zur Auswahl an. Das ist zwar nicht ganz so individuell, aber immer noch sehr praktisch für die persönliche Gestaltung.

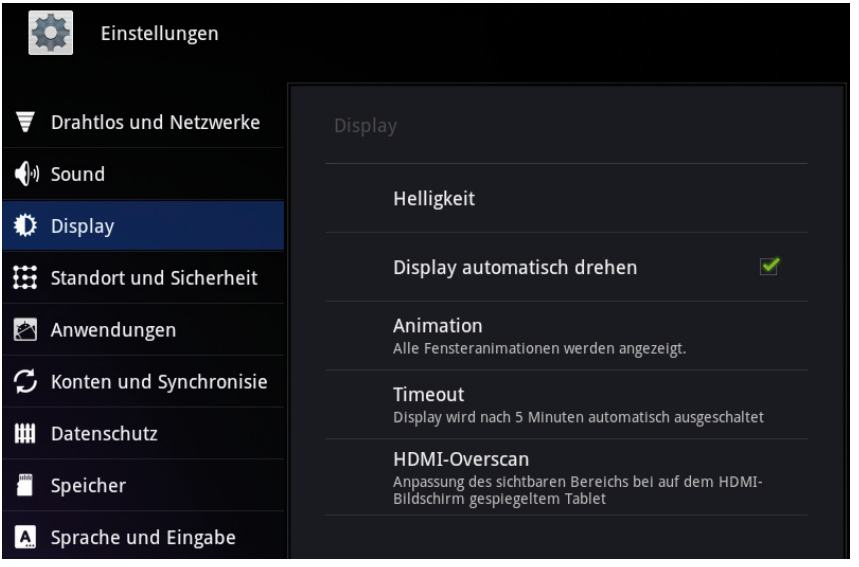


Ein Widget vergrößern/verkleinern bzw. eine andere Größe auswählen.

2.3 Steuern der Bildschirmdrehung

Ihr Tablet ist so konzipiert, dass Sie es im Hochkantformat und im Querformat nutzen können. So lesen Sie z. B. ein Buch oder eine Zeitschrift meist im Hochformat, während Sie sich Ihre Fotos oder eine Webseite in der Regel im Querformat anschauen. Sie müssen das Tablet nur kippen, und schon ändert sich die Anzeige automatisch. Manchmal ist diese Automatik aber störend oder gar nicht gewünscht, z. B. wenn Sie im Liegen etwas lesen oder das Gerät einfach nur sehr hoch oder schräg halten möchten. Dafür lässt sich die automatische Rotation schnell ein- und ausschalten.

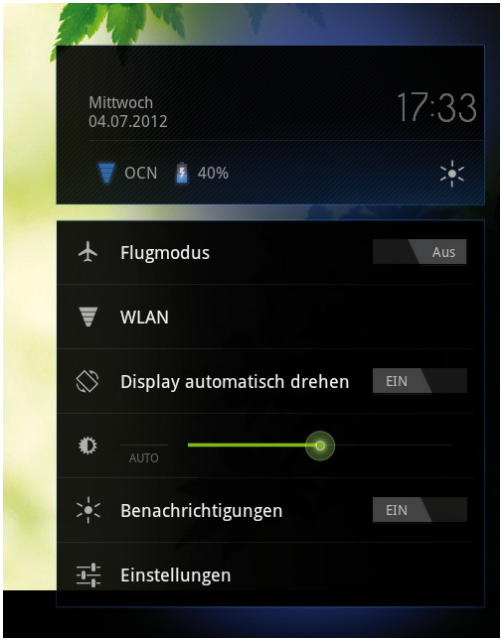
1. Der reguläre Weg führt über das Konfigurationsmenü. Tippen Sie dafür oben rechts auf *Anwendungen* und wählen Sie in diesem Fenster die Option *Einstellungen* aus. In der Gruppe *Display* lässt sich die Rotation mit *Display automatisch drehen* ein- oder ausschalten.



Die Einstellungen für die automatische Drehung.

2. Noch schneller gelangen Sie an diese Option, wenn Sie unten rechts auf die Statuszeile tippen. Es öffnet sich das Infofenster mit Datum, Uhrzeit etc. Tippen Sie rechts auf das *Einstellungen*-Symbol, und schon werden die wichtigsten Einstellungen als Schnellzugriff angezeigt. Die Option *Display auto-*

matisch drehen lässt sich hier mit einem kleinen Schiebeschalter ein- oder ausschalten.



Der Schnellzugriff über die Statuszeile.

2.4 Wichtige Programme: Apps auf dem Homescreen

Im Alltag gibt es sicherlich viele Programme bzw. Apps, die Sie täglich verwenden. Das kann der Kalender sein, Ihr E-Mail-Postfach, das Lieblingsspiel und vieles mehr. Damit Sie für diese Programme nicht jedes Mal in das Fenster *Anwendungen* gehen müssen, lassen sich auf dem Homescreen Verknüpfungen zu diesen Programmen erstellen. Dann genügt ein Fingertipp, und schon startet das Programm.

1. Wechseln Sie auf den Homescreen, auf dem Sie die App verknüpfen möchten.
2. Jetzt tippen Sie oben rechts auf die Plusschaltfläche und wählen unten das Register *Anwendungen* aus.

Christian Immler

Das inoffizielle Samsung Galaxy S4 Buch

Holen Sie alles aus Ihrem S4 heraus:
Anleitung, die besten Apps und
viele Insider-Tipps und Tricks

Inhaltsverzeichnis

1

Galaxy S4: Android-Smartphone der Extraklasse

Willkommen in der Welt der Smartphones, der Handys mit eingebautem Computer oder der Computer, die man wirklich immer bei sich haben kann. Smartphones sind heute nicht mehr nur ein Spielzeug für geschäftliche Anwender, sondern aus dem Alltag vieler Menschen kaum noch wegzudenken.

Bild 1.1: *Das Samsung Galaxy S4. (Foto: Samsung)*

Das Samsung Galaxy S4, das vom Hersteller auch als GT-I9505 bezeichnet wird, setzt eine neue Marke für die Topklasse aktueller Smartphones. Die technischen Daten – 5-Zoll-Bildschirm (12,7 cm) mit 1.080 x 1.920 Pixeln, 1,9-GHz-Quad-

Core-Prozessor, 13-Megapixel-Kamera und bis zu 42 MBit/s Download im HSPA+-Netz bzw. 100 MBit/s im LTE-Netz – liegen am oberen Rand dessen, was Smartphones zurzeit bieten können.

Bild 1.2: *Vorbereitungen zur groß angelegten Vorstellung des Samsung Galaxy S4 am 14. März 2013 auf dem Times Square in New York City. (Foto: Samsung)*

Nach Herstellerangaben wurden in den ersten vier Wochen bereits 10 Millionen Geräte vom Typ Samsung Galaxy S4 verkauft. Sein Vorgänger Samsung Galaxy SIII brauchte doppelt so lang, um diese Zahl zu erreichen, das Samsung Galaxy SII sogar fünf Monate.

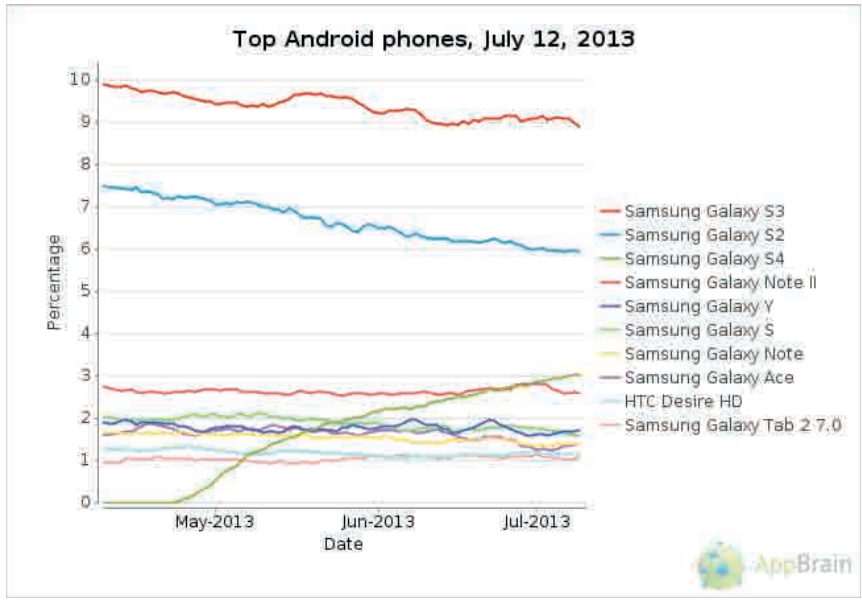


Bild 1.3: *Die am meisten verbreiteten Android-Smartphones. (Quelle: de.appbrain.com)*

Die aktuelle Statistik von *de.appbrain.com/stats/top-android-phones* zeigt, wie das Samsung Galaxy S4 in kürzester Zeit an die dritte Stelle in der Rangliste der meistgenutzten Android-Handys aufstieg. Die ersten acht Geräte dieser Liste stammen alle aus der Samsung Galaxy-Serie.

Dieses Buch zeigt, inwiefern das Samsung Galaxy S4 mehr bietet als andere Android-Smartphones, und liefert Insiderwissen und versteckte Hacks jenseits der offiziellen Handbücher.

1.1 Angesagt! – Weitere Galaxy S4-Varianten

Das Samsung Galaxy S4 gehört, seitdem es auf dem Markt ist, zu den angesagtesten Smartphones überhaupt. Da wundert es nicht, dass Samsung wie schon beim S3 die Bekanntheit des Namens nutzt und weitere Geräte mit der Bezeichnung S4 auf den Markt bringt.

1.1.1 Samsung Galaxy S4 Google Edition

Das Samsung Galaxy S4 erscheint zusätzlich in einer Google Edition, auch als Nexus Edition bezeichnet, mit reinem Android – allerdings vorerst nur in den USA. Diese Variante unterscheidet sich vom Original-S4 lediglich in der installierten Software. Die Google Edition verwendet sogenanntes Stock Android ohne TouchWiz-Oberfläche und ohne einige der vorinstallierten Apps. Auch fehlen der verbesserte Kalender und der Notizblock sowie die Erweiterungen in der Kamerasoftware.

Ein Vorteil der Google Edition ist die schnelle Versorgung mit Updates. Ähnlich wie auf den Nexus-Smartphones kümmert sich Google selbst um die Updates, und Nutzer müssen nicht warten, bis Samsung eine neue Android-Version an die TouchWiz-Oberfläche und die vorinstallierten Systemmodifikationen angepasst hat. Außerdem soll der Bootloader der Google Edition-Geräte entsperrt sein, sodass man problemlos alternative Android-ROMs oder auch andere Betriebssysteme, wie z. B. Ubuntu, installieren kann.

1.1.2 Samsung Galaxy S4 Advanced

Bis jetzt nur in Südkorea wird eine besonders schnelle Geräteversion unter dem Namen Samsung Galaxy S4 Advanced angeboten, die den neuen Datenübertragungsstandard LTE Advanced mit bis zu 1.000 MBit/s unterstützt, sich sonst aber nicht vom Original unterscheidet. Samsung plant, dieses Modell auch in anderen Ländern anzubieten, sobald dort die technische Infrastruktur für LTE Advanced zur Verfügung steht. In Deutschland testet Vodafone bereits den Netzausbau für LTE Advanced.

1.1.3 Samsung Galaxy S4 Active

Unter dem Namen Galaxy S4 Active oder auch Samsung I9295 liefert Samsung eine besonders robuste Version des Smartphones, das nach IP67-Zertifizierung staubdicht ist und eine halbe Stunde lang unbeschadet in ein Meter tiefem Wasser liegen kann. Dazu wurde die Chromumrandung des Geräts durch einen Kunststoffrand mit speziellen Dichtungen an den Anschlüssen für USB und Kopfhörer sowie an den Tasten ersetzt. Die von anderen Outdoor-Handys bekannten Gummiwülste gibt es hier nicht. Samsung erwähnt auch keine besondere Stabilität bei Stürzen und Stößen.

Einen weiteren sichtbaren Unterschied stellen die drei mechanischen Tasten für Menü, Home und Zurück unterhalb des Bildschirms dar. Das Originalgerät Samsung Galaxy S4 hat nur eine Home-Taste und zwei Sensortasten, die sich aber unter Wasser oder bei starkem Regen nicht bedienen lassen. Für Fotos in solchen Umgebungen gibt es einen speziellen Aqua-Modus, in dem die Lautstärketasten zur Kamerasteuerung verwendet werden können. Allerdings hat die Kamera nur 8 Megapixel Auflösung im Gegensatz zu den 13 Megapixeln des Originals.

Bild 1.4: *Links: Samsung Galaxy S4 Active – rechts: Samsung Galaxy S4 Mini. (Fotos: Samsung)*

1.1.4 Samsung Galaxy S4 Mini

Das Samsung Galaxy S4 Mini oder auch Samsung I9195 soll all die ansprechen, denen das »echte« Samsung Galaxy S4 zu groß oder einfach zu teuer ist. Samsung hat hier an mehreren Komponenten gespart: am Bildschirm, am Speicher, an der Kamera und am Prozessor.

Technische Daten	Samsung Galaxy S4	Samsung Galaxy S4 Mini
Maße	137 x 70 x 8 mm	125 x 61 x 9 mm
Gewicht	129 g	107 g
Bildschirmdiagonale	12,7 cm (5 Zoll)	10,9 cm (4,3 Zoll)
Bildschirmauflösung	1.080 x 1.920 Pixel	960 x 540 Pixel
Prozessor	Quad-Core	Dual-Core
Taktfrequenz	1,9 GHz	1,7 GHz
Interner Speicher	16/32 GByte	8 GByte
Akku	2.600 mAh	1.900 mAh
Kamera	13 Megapixel	8 Megapixel

Weiterhin fehlen dem Samsung Galaxy S4 Mini die Gestenerkennung sowie einige der Sensoren – Thermometer, Barometer, Hygrometer. Betriebssystem, Benutzeroberfläche und die von Samsung mitgelieferten Apps sind bei beiden Modellen gleich.

1.1.5 Samsung Galaxy S4 Zoom

Das Samsung Galaxy S4 Zoom ist eine interessante Mischung aus Smartphone und Kompaktkamera mit 16 Megapixeln und zehnfach optischem Zoomobjektiv. Mit dem originalen Samsung Galaxy S4 hat es nur noch den Namen gemeinsam, die Hardwareausstattung entspricht weitgehend dem Samsung Galaxy S4 Mini mit 4,3-Zoll-Bildschirm, 960 x 540 Pixeln Auflösung und 1,5-GHz-Dual-Core-Prozessor.

Bild 1.5: *Samsung Galaxy S4 Zoom. (Fotos: Samsung)*

Im Gegensatz zum Vorgänger, der Galaxy Camera, kann man mit dem Samsung Galaxy S4 Zoom auch telefonieren. Zum besseren Fotografieren gibt es einen echten Kameraauslöser sowie einen Drehring zum Zoomen am Objektiv, der auch zum Einschalten der Kamera während eines Telefongesprächs verwendet wird. Wie bei vergleichbaren Kompaktkameras sind ein optischer Bildstabilisator und ein Xenon-Blitz eingebaut. Für Videotelefonie gibt es auf der Vorderseite noch eine kleine Kamera mit 2 Megapixeln.

1.2 Akku, Ladegerät und microSIM-Karte

Aufgrund internationaler Sicherheitsvorschriften müssen Akkus immer im leeren Zustand verschickt werden. Das Samsung Galaxy S4 verwendet wie alle aktuellen Android-Handys ein MicroUSB-Ladegerät. Diese Ladegeräte sind beliebig zwischen den Handys austauschbar. Wer mehrere Geräte nutzt, braucht nicht immer mehrere Ladegeräte mit sich herumzutragen. Seit der Vereinheitlichung der Ladegeräte für alle Smartphones außer dem iPhone kann man bequem ein Ladegerät fest am Schreibtisch oder in der Küche deponieren, ein weiteres am Arbeitsplatz oder ähnlich.

Das mit dem Samsung Galaxy S4 mitgelieferte Ladegerät liefert einen Ladestrom von 2.000 mA und ist damit stärker als die meisten Ladegeräte einfacher Handys. Nach etwa einer Stunde ist der Akku so weit voll, dass sich das Samsung Galaxy S4 problemlos in Betrieb nehmen lässt.

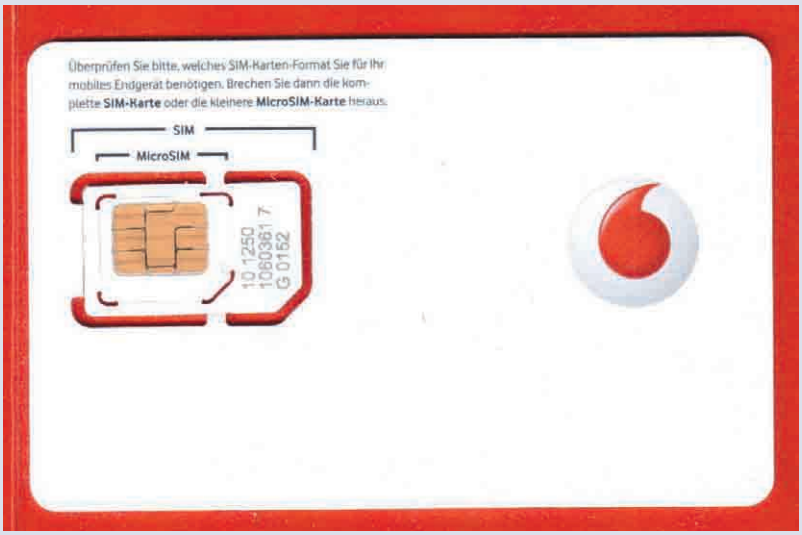
1. Drücken Sie zum Erststart länger (etwa eine Sekunde) auf den Einschalter. Der Bildschirm wird etwas heller, und nach kurzer Zeit erscheint das Logo des Handyherstellers.

2. Als Erstes nach dem Einschalten müssen Sie wie auf jedem Handy die PIN Ihrer SIM-Karte eingeben. Je nach SIM-Karte müssen Sie eventuell vorher noch Ihren Netzanbieter manuell aus einer Liste auswählen.

3. Die meisten Funktionen des Samsung Galaxy S4 lassen sich im WLAN auch ohne SIM-Karte nutzen. Ist keine SIM-Karte eingelegt, wird die PIN-Eingabe automatisch übersprungen. Anhand der SIM-Karte wird bei der Ersteinrichtung sofort auch ein Internetzugang über diese SIM-Karte eingerichtet, der allein durch Hintergrunddienste bereits Kosten verursachen kann (siehe dazu weiter unten im Kapitel »2.7 Günstig, aber gut: Tipps zum Handytarif«. Wenn Sie keinen Internettarif auf Ihrer SIM-Karte haben oder sich nicht sicher sind, führen Sie die Erstinstallation per WLAN durch und stecken dazu keine SIM-Karte in das Gerät.

microSIM-Karte

Das Samsung Galaxy S4 verwendet microSIM-Karten und nicht wie die meisten Android-Smartphones die typischen miniSIM-Karten. Wer sein Samsung Galaxy S4 nicht direkt mit einem Mobilfunkvertrag kauft, muss also darauf achten, von seinem Provider eine microSIM-Karte zu bekommen. Alle vier Netzbetreiber sowie auch immer mehr Discounter liefern inzwischen sogenannte Kombi-SIM-Karten aus. Mehrere Stanzlinien ermöglichen es hier, den SIM-Chip entweder in Form einer klassischen miniSIM oder als microSIM aus dem Kartenträger herauszudrücken.



Die microSIM-Karten sind einfach nur kleiner, die Kontakte aber gleich angeordnet und elektronisch voll kompatibel zu normalen miniSIM-Karten. Lediglich der interne Speicher wurde in der Spezifikation der microSIM-Karte um 50 % vergrößert. Im Zubehörhandel werden Adapter angeboten, in die man eine microSIM-Karte einklemmt und diese dann in jedem Handy oder auch in USB-Surfsticks nutzen kann.

Umgekehrt findet man im Internet Anleitungen und Schneidevorlagen, um normale miniSIM-Karten auf die Größe einer microSIM zurechtzustutzen. Der eigentliche Chip in den SIM-Karten liegt genau unter der Kontaktfläche und kann, sofern man mit einem scharfen Messer sauber schneidet und die SIM-Karte dabei nicht zerplatzt, nicht beschädigt werden. Üben Sie also am besten einmal mit einer abgelaufenen oder einer kostenlosen Promo-SIM-Karte, bevor Sie das Messer an der echten SIM-Karte ansetzen. Für weniger Mutige gibt es im Handyzubehörhandel einfache Stanzmaschinen für etwa 10 Euro, mit denen man kaum etwas falsch machen kann.

Prevezanos / Rehberg

Android XL-Edition

Apps · Tuning · Sicherheit

Inhaltsverzeichnis

2 Android personalisieren

Ihr Smartphone ist Ihr ganz persönlicher Assistent. Sie haben ihn immer dabei, er wacht über Ihre Kontakte, Termine und Konversationen, und natürlich speichert er alle Ihre wichtigen Fotos und Musikstücke. Da ist es nur selbstverständlich, dass Sie Ihr Smartphone auch an Ihren ganz persönlichen Geschmack anpassen möchten. Verändern Sie den Hintergrund, die Farben, die Töne oder auf Wunsch auch gleich die komplette Benutzeroberfläche. Android ist ähnlich stark anpassbar wie ein Desktopcomputer. Es gibt kaum etwas, das sich nicht anpassen lässt, sodass Sie nach ein wenig Ausprobieren und Umbauen Ihr ganz persönliches Android gestalten. Dieses Kapitel zeigt Ihnen, wie Sie die wichtigsten Elemente an Ihre Wünsche anpassen.

2.1 Hintergrundbild ändern

Möchten Sie das Aussehen Ihres Smartphones schnell und unkompliziert verändern, geht das natürlich über das Hintergrundbild. So sehen Sie immer Ihr Lieblingsmotiv auf dem Bildschirm, und jeder erkennt sofort, dass dies Ihr Smartphone ist. Dabei beherrscht Android gleich mehrere Arten von Hintergrund, die den Bildschirm nicht nur schmücken, sondern richtig interessant machen.

1. Betätigen Sie auf der Startseite des Smartphones die Taste *Menü* und wählen Sie aus dem sich öffnenden Menü den Punkt *Hintergrund* beziehungsweise *Hintergrundbild* aus.

2. Nun öffnet sich eine Liste mit drei verschiedenen Arten von Hintergrundbild.

   Mit dem Punkt *Hintergrundbild* beziehungsweise *Hintergrundbildgalerie* wählen Sie aus den vorinstallierten Hintergrundbildern eines aus.

   Über den Punkt *Galerie* wählen Sie ein eigenes Foto von der Speicherkarte aus und gestalten damit ein mehrseitiges Panorama.

   Wählen Sie die Option *Live-Hintergründe* aus, wird Ihr Bildschirm zu einer Animation mit vielen Sonderfunktionen.

Bild 2.1: Die Option *Hintergrundbild* und die Art des Bilds auswählen.

3. Besonders schnell geht es, wenn Sie die Option *Hintergründe* beziehungsweise *Hintergrundbildgalerie* auswählen. Sie gelangen in eine Liste mit vorinstallierten Bildern, tippen das gewünschte an und betätigen die Schaltfläche *Hintergrundbild festlegen*.

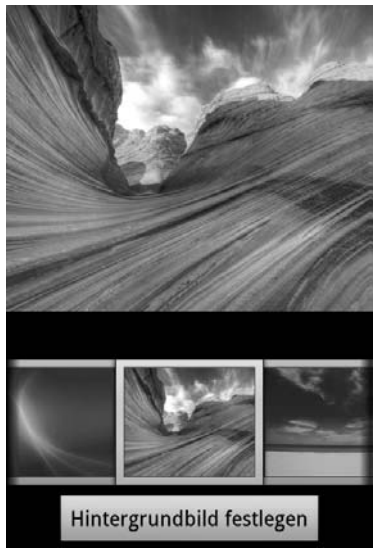


Bild 2.2: Einen Standardhintergrund festlegen.

4. Wollen Sie lieber ein eigenes Foto verwenden, gelangen Sie über die Option *Galerie* in die Fotogalerie Ihres Smartphones. Wählen Sie dort den Ordner und das gewünschte Foto aus.

Bild 2.3: Ein eigenes Foto auswählen.

5. Jetzt wird das Foto mit einem Rahmen angezeigt, sodass Sie den gewünschten Ausschnitt auswählen können. Per Standard wird das Foto so angepasst, dass es sich über mehrere Seiten des Homescreens erstreckt. Dann bewegt es sich beim Blättern sanft mit.
6. Bei einigen Modellen können Sie aber auch festlegen, ob das Bild als Panorama zurechtgeschnitten oder als einzelne Bildschirmseite verwendet werden soll. Dann sehen Sie auf jeder Seite des Homescreens dasselbe Bild mit demselben Ausschnitt.

Bild 2.4: Den Bildausschnitt sowie die Fotobreite festlegen.

7. Möchten Sie lieber eine Animation als Hintergrund, wählen Sie am Anfang die Option *Live-Hintergründe* aus. Tippen Sie in der Liste auf das gewünschte Bild, um eine Vorschau zu erhalten.
8. Einige Live-Hintergründe lassen sich über die Schaltfläche *Einstellungen* anpassen, z. B. in der Art der Animation oder der anzuzeigenden Inhalte. Mit der Schaltfläche *Hintergrundbild festlegen* speichern Sie diesen Live-Hintergrund.

Bild 2.5: Animierte Hintergründe mit Anzeigeoptionen.

2.2 Homescreen verwalten

Der Hauptbildschirm wird bei Android als Homescreen bezeichnet und stellt die oberste Ebene der Benutzeroberfläche dar. Er ist immer sichtbar, wenn Sie gerade nichts mit Ihrem Smartphone tun, und zeigt die wichtigsten Standardinformationen an. Auf dem Homescreen lassen sich Verknüpfungen zu Ihren bevorzugten Programmen erstellen, Sie können Fotos einblenden, Mini-Anwendungen – sogenannte Widgets – starten und vieles mehr.

Der Homescreen ist also durchaus mit der Desktopoberfläche eines PCs vergleichbar. Unter Android gibt es aber nicht nur einen Homescreen, sondern gleich mehrere. Sie können also mehrere Bildschirme individuell gestalten und nach Belieben zwischen diesen wechseln.

- Wischen Sie mit dem Finger über den Touchscreen und schieben Sie den aktuellen Homescreen nach rechts oder links, um zur nächsten Seite zu gelangen.
- Anhand von Punkten, Nummern oder ähnlichen Symbolen wird Ihnen dabei immer angezeigt, auf welchem Homescreen Sie sich gerade befinden.

- Jeder Homescreen besitzt im unteren Bereich eine gleichbleibende Funktionsleiste mit den Grundfunktionen des Telefons, z. B. *Telefon*, *Nachrichten*, *Kontakte*, *Anwendungen* usw. Die Anwendungen variieren je nach Hersteller.

Bild 2.6: Die Android-Homescreens 1, 2 und 3 (Beispiel: Samsung Galaxy SII).

Während Sie sich in den Menüs oder Programmen Ihres Smartphones bewegen, können Sie jederzeit über die Taste *Home* zurück zum Homescreen gelangen. Allerdings sind dabei die Reihenfolge und die Priorität der Screens nicht festgelegt. Einige Hersteller zählen die Screens von links nach rechts durch, und Sie springen mit der *Home*-Taste immer auf Seite 1. Bei anderen Herstellern ist der mittlere Screen der Hauptscreen, und die anderen Seite sind rechts und links davon angeordnet.

2.2.1 Vorgegebene Homescreen-Seiten bearbeiten

Android sieht standardmäßig fünf Homescreens vor. Viele Hersteller passen die Oberfläche für ihre Modelle an und bieten die Möglichkeit, nicht genutzte Seiten zu entfernen oder weitere Seiten hinzuzufügen. So erstellen Sie z. B. bei Samsung-Modellen bis zu sieben Screens oder reduzieren das Ganze auf nur drei Seiten. Haben Sie aus dem Google Play Store eine neue Oberfläche installiert, stehen Ihnen noch mehr Möglichkeiten offen.

1. Tippen Sie auf dem Homescreen die *Menü*-Taste an, sodass sich das Optionsmenü öffnet.
2. Bietet Ihr Smartphone eine Anpassung der Homescreen-Seiten an, finden Sie dort die Option *Bearbeiten*.

Bild 2.7: Die Homescreen-Seiten bearbeiten.

3. Neue Homescreens erstellen Sie mithilfe des großen Pluszeichens oder über den Menüpunkt *Seite/Screen hinzufügen*.
4. Nicht genutzte Seiten tippen Sie mit dem Finger an und halten diese einen Moment fest. Nun wird die Seite beweglich und kann in den Mülleimer geschoben werden.

Christian Bleske

Java für Android

Native Android-Apps programmieren

Vorwort

Liebe Leserinnen und Leser,

egal welche Studie man sich derzeit ansieht, ein Trend ist klar erkennbar. Die Nutzung von mobilen Geräten nimmt weiter rasant zu. Ein Gewinner steht dabei ganz klar fest: Googles Android. Das (mobile) Betriebssystem hat in Deutschland derzeit einen Marktanteil von über 60 und weltweit von ca. 75 Prozent. Tendenz weiter steigend. In diesem Sinne ist die Beschäftigung mit dem Thema also weiterhin zukunftsweisend.

Vor ca. einem Jahr erschien die erste Auflage von »Java für Android«. An dieser Stelle darf ich Sie zur zweiten, überarbeiteten Auflage begrüßen. Obwohl es nur einen kleinen Versionssprung von 4.0 auf 4.2 gab, hat sich für die Entwickler einiges getan. Gerade im Bereich der Entwicklung von Apps für Android wurde mit der Veröffentlichung der neuen ADT-Werkzeuge der Komfort deutlich erhöht. Der Einstieg in die Thematik dürfte somit noch einfacher fallen.

Eigentlich müsste dieses Buch »Java & XML für Android« heißen. Warum? Nun, wenn Sie dieses Buch lesen, dann werden Sie feststellen, dass die Entwicklung einer Android-App neben Java auch viel XML erfordert. Aber das wäre ein blöder Titel, und so heißt das Buch eben, wie es heißt.

Dieses Buch enthält über 50 Apps (Download unter *www.buch.cd*). Jedes Thema wurde als eigenständige App umgesetzt, die auf ein Android-Smartphone übertragen und dort ausgeführt werden kann. Nutzen Sie diesen Fundus. Wenn Sie eine Idee für eine App haben, dann schauen Sie doch einmal unter den Beispielen, ob Sie nicht eines als Basis für Ihre eigene App verwenden können. Wie sagt man so schön: »Bitte bedienen Sie sich!«. Zusätzlich zu den bereits aus der ersten Auflage bekannten Kapiteln habe ich in Kapitel 24 auch ein neues Beispiel zur App-Programmierung hinzugefügt, welches dazu dienen soll, Ihnen die Entwicklung einer App vom Beginn bis zum Ende an einem einfachen Beispiel komplett zu demonstrieren. Komplett deshalb, weil es nicht nur der Demonstration einer bestimmten Technik (also beispielsweise eines Controls) dient, sondern selbst etwas Logik beinhaltet. Es handelt sich um ein Zahlenratespiel.

Wenn Ihnen nach dem Lesen des Buches noch Ideen zur eigenen App fehlen oder Sie ein größeres Beispiel suchen, dann öffnen Sie doch einmal den Google Play Store und betrachten Sie die »Wuppertal App«. Diese App ist übrigens zur selben Zeit entstanden wie die erste Auflage des Buchs.

Menschen machen Fehler. Natürlich schließt das mich als Autor ein. Die Wahrscheinlichkeit ist also groß, dass Sie auch in der zweiten Auflage auf den einen oder anderen Fehler stoßen werden. Behalten Sie ihn! Nein, natürlich nicht! Wenn Sie einen Fehler finden oder sich eine andere Unklarheit in Bezug auf das Buch ergibt, so bin ich unter cb.2000@hotmail.de für Sie erreichbar.

Auch für Kritik (Beim nächsten Mal wird alles noch besser!) oder Lob bin ich Ihnen dankbar. Schreiben Sie mir, ich werde versuchen, jede Mail zu beantworten.

Herzlich bedanken möchte ich mich bei den Mitarbeiterinnen und Mitarbeitern des Franzis Verlags und hier im Besonderen bei Herrn Markus Stäuble (Programmleitung Web- und Smartphoneprogrammierung).

Ein besonders großer Dank geht an meine Frau Sanela. Sie ermöglicht es mir, dass ich Zeit zum Schreiben finde.

Christian Bleske im Januar 2013

Inhaltsverzeichnis

2 Eclipse & ADT-Plugin

Im ersten Kapitel wurden mit der Installation des Java- und des Android-SDKs die Grundlagen für die Entwicklung von Android-Apps gelegt. Allein mit diesen beiden Frameworks ist eine Entwicklung von Apps zwar möglich, aber äußerst umständlich. Um schneller Ergebnisse erzielen zu können und auch um den Entwicklungsprozess zu vereinfachen, ist die Verwendung einer integrierten Entwicklungsumgebung unumgänglich. Dabei muss es nicht unbedingt *Eclipse* sein, auch die Verwendung von beispielsweise *NetBeans* als IDE (Integrated Development Environment) ist möglich. Für Eclipse spricht allerdings, dass Google für diese Entwicklungsumgebung eine Erweiterung (Plugin) mit dem Namen ADT-Plugin programmiert hat. Dieses ADT-Plugin sorgt dafür, dass aus der IDE Eclipse quasi eine Android-IDE wird. So steckt beispielsweise ein Designer für die Entwicklung von grafischen Benutzeroberflächen in der Erweiterung und es sind zusätzlich Werkzeuge für die Fehlersuche integriert. Auch gibt es nach der Installation Vorlagen für die Erstellung eines App-Projekts.

In diesem Kapitel erfahren Sie nicht nur, wie Eclipse und das ADT-Plugin installiert werden. Es werden auch die Bestandteile der IDE, die wichtigsten Werkzeuge (z. B. zur Fehlersuche) und die Struktur sowie die Bestandteile eines Android-Projekts erläutert.

Hinweis
Der Abschnitt über die Installation von Eclipse und ADT-Plugin ist nur dann interessant, wenn man nicht das ADT Bundle verwendet. Dies ist beispielsweise dann der Fall, wenn Sie Eclipse auch noch für andere (Java-)Projekte verwenden möchten.

2.1 Download und Installation von Eclipse

Bei Eclipse handelt es sich um eine Entwicklungsumgebung, die von der Open-Source-Community entwickelt wird. Eclipse kann nicht nur für die Entwicklung von Android-Apps verwendet werden. Hauptsächlich wird Eclipse sogar für die Entwicklung von Java-Anwendungen verwendet. Egal ob es sich dabei um Java-Anwendungen für den Desktop oder Web-Anwendungen handelt. Durch die besondere Architektur von Eclipse, die auf Erweiterungen (Plugins) basiert, ist die IDE sehr flexibel konfigurierbar und somit in einer Vielzahl von unterschiedlichen Projekten zu gebrauchen. Mit Eclipse können darüber hinaus auch PHP- oder C- und C++-Anwendungen entwickelt werden. Neben den Möglichkeiten zur Erweiterung ist ein weiterer Vorteil der IDE, dass sie für eine Vielzahl von Plattformen (von Windows über diverse Linux-Versionen bis hin zu Mac OS X) verfügbar ist.

Die Verwaltung und Verteilung von Eclipse wird über die Webseite *eclipse.org* durchgeführt. Auf dieser Webseite hat man die Möglichkeit, die vielen verschiedenen Varianten der Entwicklungsumgebung herunterzuladen. Außerdem werden dort auch Neuigkeiten zu Änderungen und Erweiterungen von Eclipse vorgestellt. Die Downloads sind unter *http://www.eclipse.org/downloads/* zu finden.

Öffnet man diese Seite, so ist dort eine Vielzahl an unterschiedlichen Eclipse-Versionen aufgelistet.

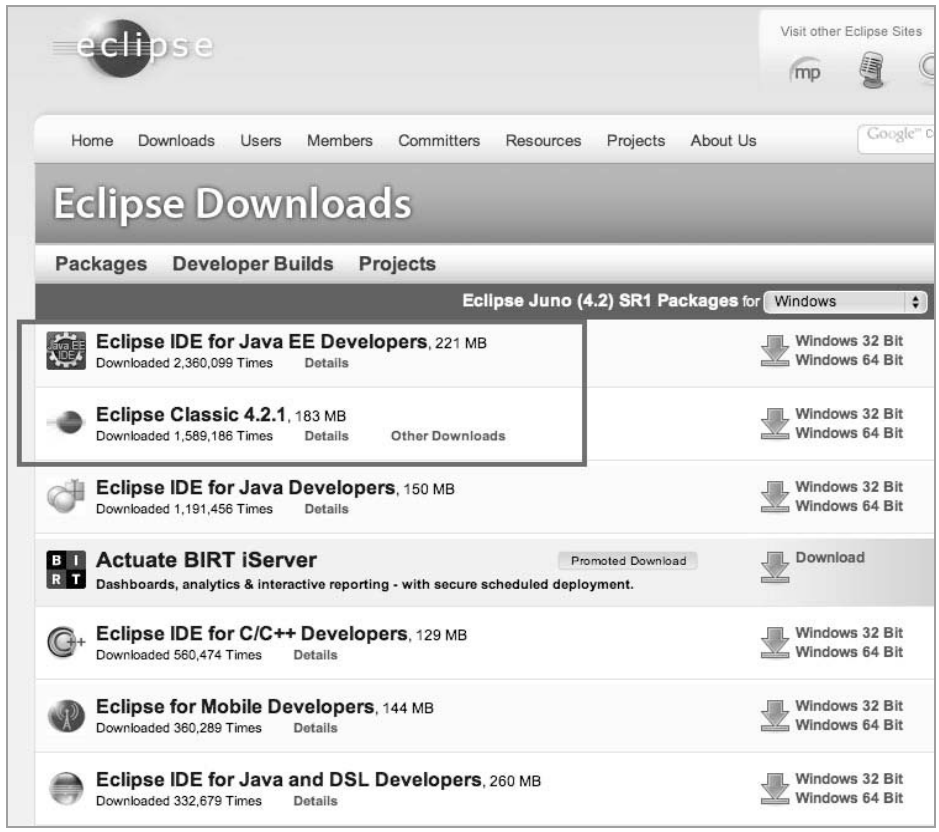


Bild 2.1: Der Download von Eclipse

Welche aber ist die richtige für die Entwicklung von Android-Apps? Von den aufgeführten Versionen eignen sich nicht alle für die Entwicklung von Android-Apps. Empfohlen wird die Verwendung von *Eclipse Classic*. Innerhalb dieser Variante von Eclipse lässt sich das ADT-Plugin installieren. Man sollte allerdings darauf achten, dass die Versionsnummer größer 3.5 ist. Mit den neueren Versionen (3.8, 4.2) gibt es hin und wieder Probleme bei der Installation des Plugins. Mit Version 3.6 (Codename *Helios*) scheint das bisher eher weniger der Fall gewesen zu sein. Für die Android-Entwicklung spielt es keine Rolle, ob 3.6, 3.7 oder eine neuere Version verwendet wird. Um *Eclipse Helios* herunterzuladen, muss man allerdings dem Link *Other Downloads*

folgen. Erst dann kommt man zu einer Webseite, von der diese Variante geladen werden kann. Übrigens finden Sie hier auch den Download für die Mac-OS-X-Versionen.

Hinweis
Eine Alternative bei Problemen kann auch der Download der *Eclipse IDE for Java EE Developers* sein. Der Download ist zwar etwas größer, in dieser Version sind aber alle Bibliotheken enthalten, so dass es bei der Installation des ADT-Plugins zu keinen Problemen kommen sollte.

Nach dem Download, der durchaus einige Zeit in Anspruch nehmen kann, folgt die Installation. Diese fällt bei Eclipse sehr kurz aus und beschränkt sich auf das Entpacken des ZIP-Archivs und das Erstellen einer Verknüpfung zur *Eclipse.exe* für den Desktop von Windows. Im Anschluss kann Eclipse das erste Mal gestartet werden.

Zuerst wird der Eclipse-Splash-Screen eingeblendet. Während des Ladens wird noch kurze Zeit ein Dialog (Workspace Launcher) angezeigt.

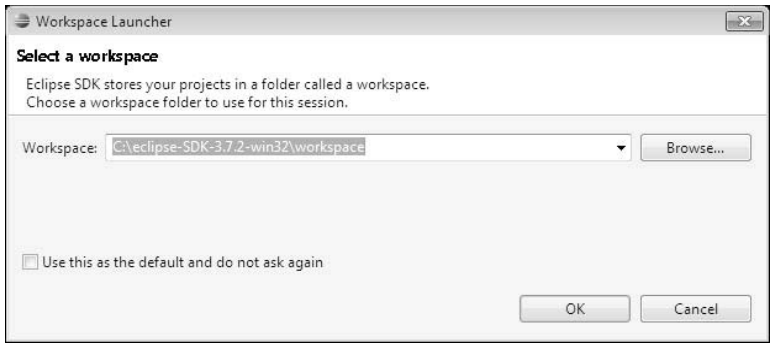


Bild 2.2: Konfiguration des Workspace-Verzeichnisses

Die Eclipse-IDE verwaltet alle Projekte in einem eigenen Ordner, der den Namen *Workspace* trägt. Direkt nach der Installation ist dieser Ordner natürlich noch nicht vorhanden. Aus diesem Grund muss der Anwender nun festlegen, wo der Ordner angelegt werden soll. Bewährt hat sich, den Workspace-Ordner unterhalb des Eclipse-Ordners anzulegen. Aber auch andere Konstellationen sind natürlich möglich. Damit man nicht nach jedem Start von Eclipse erneut gefragt wird, sollte die Option *Use this as the default* ... aktiviert werden.

2.2 Eclipse – kurze Einführung

Nachdem Eclipse vollständig geladen wurde, wird der Welcome-Dialog angezeigt.

Im Welcome-Dialog werden vier Optionen angeboten: *Overview*, *Tutorials*, *Samples* und *What's New*. Hier erhält man Informationen zur Eclipse-Plattform, Grundlagen zur Entwicklung von (u. a.) Java- und SWT-Anwendungen mit Eclipse, dazu einige Programmierbeispiele, und man erfährt, was es Neues über Eclipse zu wissen gibt. Alle diese Punkte helfen Ihnen, die Eclipse-Plattform und die Bedienung der IDE besser zu verstehen. Zur Entwicklung von Android-Apps ist dieses Wissen aber nicht zwingend erforderlich. Man kann den Dialog über das Schließen-Symbol (links oben) beruhigt verlassen. Erst wenn der Welcome-Dialog geschlossen wurde, zeigt sich die Entwicklungsumgebung.

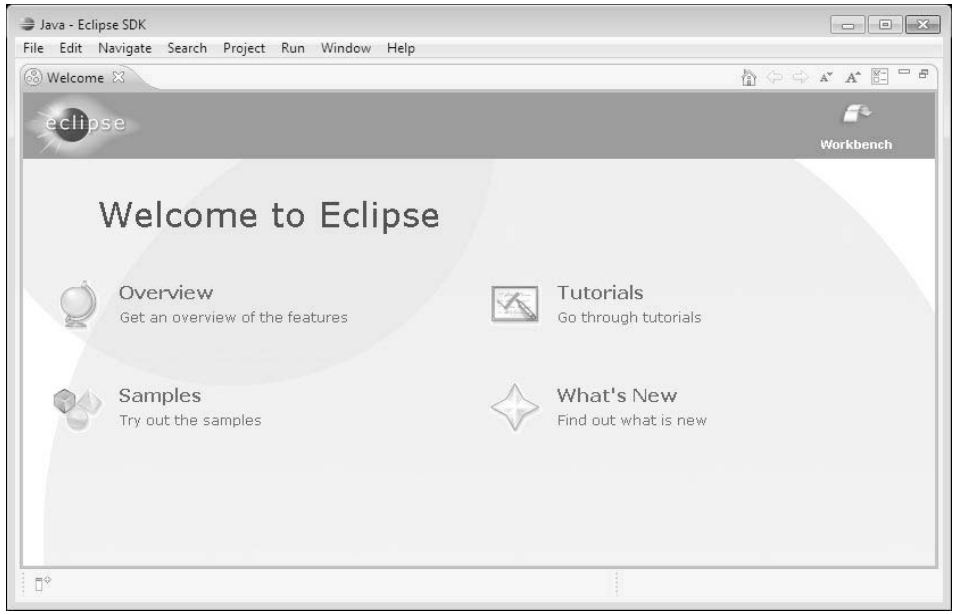


Bild 2.3: Welcome-Dialog von Eclipse

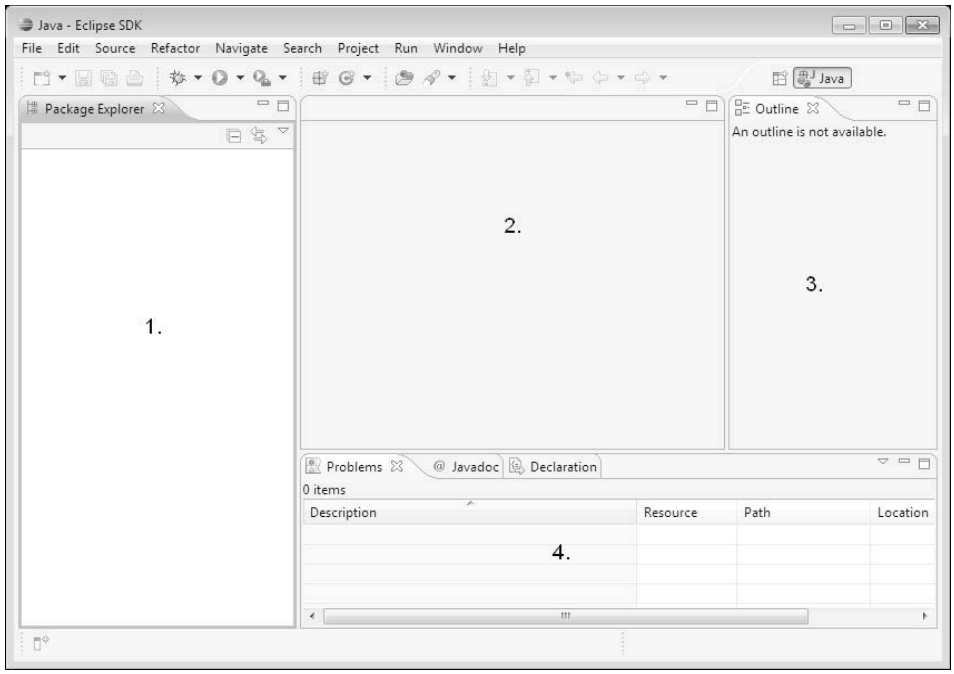


Bild 2.4: Die Eclipse-IDE

Neben der Menü- und Buttonleiste werden vier »Bereiche« angezeigt, mit welchen Sie später arbeiten werden. In Bild 2.4 wurden diese Bereiche durchnummeriert. Der Bereich mit der Nummer 1 ist der sogenannte *Package Explorer*. In diesem wird ein Projekt hierarchisch angezeigt. Dem *Package Explorer* ist in diesem Kapitel ein eigener Abschnitt (2.4.1) gewidmet. Daher später mehr hierzu. Im Bereich mit der Nummer 2 werden im Laufe der Entwicklung eines Projekts verschiedene Ansichten zu sehen sein. Beispielsweise wird in diesem Bereich später der Editor für den Java-Code angezeigt, aber auch die Anzeige des XML-Editors (mit diesem werden die Oberflächen von Android-Apps erstellt) ist hier zu finden. Auch der grafische Designer für Benutzeroberflächen wird hier eingeblendet. In den Bereichen 3 und 4 können unterschiedliche Fenster mit Statusinformationen angezeigt werden. Wenn Sie im Bereich 2 ein Objekt selektiert haben, so können im Bereich 3 oder 4 Detailinformationen zum ausgewählten Objekt angezeigt werden. Da die Fenster nicht fest »installiert« sind, können sie nach eigenem Geschmack verschoben und in einem der Bereiche wieder verankert werden. Sie können das ruhig ausprobieren. Die Register können per Drag & Drop aus einem Bereich in einen anderen verschoben werden.

Werfen Sie nun einen Blick in die einzelnen Menüpunkte. Unterhalb des *File*-Menüs finden Sie Punkte, um neue Projekte anzulegen (*New*) oder auch vorhandene Projekte zu importieren (*Import...*). Viele der Punkte sind selbsterklärend. Alle anderen werden bei Bedarf vorgestellt. Das *Edit*-Menü enthält die bekannten Funktionen zum Markieren, Kopieren, Einfügen und Suchen von (Text-)Objekten. Unter dem Punkt *Navigate* befinden sich Optionen zur Navigation in Projekten. Es folgt das *Search*-Menü

und anschließend *Project*. Im *Project*-Menü finden Sie Punkte, um das aktuelle Projekt zu erstellen und zu bearbeiten. Im folgenden *Run*-Menü befinden sich Menüeinträge zur Fehlersuche und zum Ausführen von Projekten. Das *Window*-Menü erlaubt den Zugriff auf weitere Fenster bzw. Dialoge. Wichtig ist hier vor allem der Menüeintrag *Show View*, über welchen Sie weitere Fenster der Entwicklungsumgebung öffnen können. Beispielsweise können Sie via *Show View* das *Problems*-Register anzeigen lassen. Ein weiterer wichtiger Dialog wird über den Eintrag *Preferences* aufgerufen.

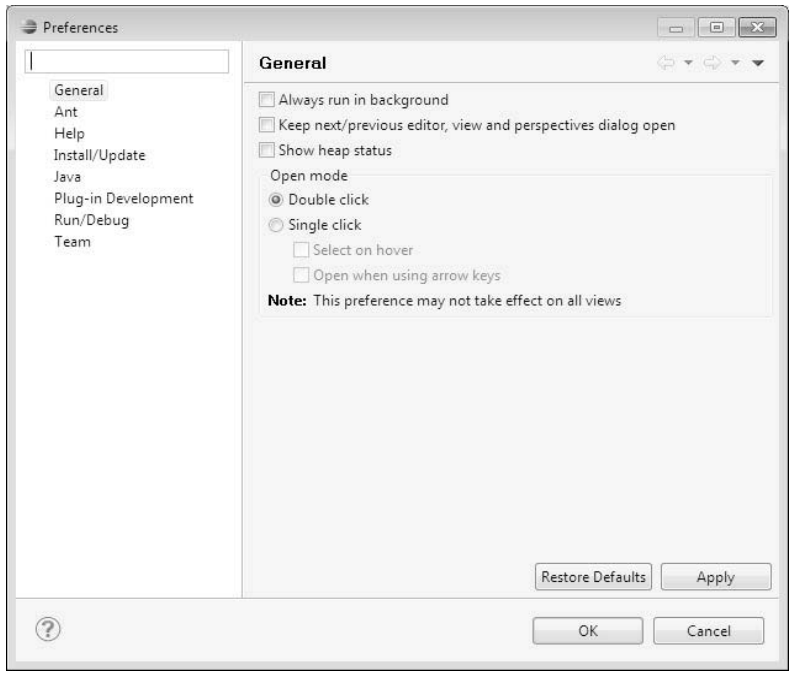


Bild 2.5: Konfiguration der Eclipse-IDE

Im *Preferences*-Dialog wird die Konfiguration von Eclipse vorgenommen. Im Abschnitt *General* befinden sich Einstellungen, welche die Eclipse-IDE direkt betreffen. Hier kann man u. a. das Aussehen der Entwicklungsumgebung konfigurieren, aber auch Änderungen am Text-Editor lassen sich hier vornehmen. Erst nach der Installation des ADT-Plugins findet sich hier ein Punkt mit dem Namen *Android*. Über diesen können bestimmte Bereich des ADT-Plugins und somit der Android-Entwicklung geändert werden. Beispielsweise lässt sich hier einstellen, in welchem Verzeichnis sich die Android-Schlüsseldateien für das Debugging befinden.

Anmerkung

Android-Apps benötigen, um auf einem Android-Smartphone ausgeführt werden zu können, eine Signierung. Zur Erstellung dieser Signierung wird eine Schlüsseldatei *Dateiname.keystore* verwendet. Es gibt eine Schlüsseldatei, mit der eine Debug-Signierung (*debug.keystore*) erzeugt werden kann, deren Laufzeit maximal ein Jahr beträgt. Ein Schlüssel zur Signierung für eine reguläre Veröffentlichung hat dagegen eine

Laufzeit von mindestens 20 Jahren. Diese Schlüsseldateien werden noch an einigen Stellen im Buch thematisiert.

Neben den Punkten *General* und *Android* findet man in diesem Bereich u. a. Konfigurationsmöglichkeiten zu den Themen *Installation und Update*, *Java*, *Plugin-Entwicklung* sowie *XML*.

Ein weiterer wichtiger Punkt ist im Menü *Help* zu finden. Dort gibt es die beiden Einträge *Check for Updates* und *Install New Software...* . Der Menüeintrag *Check for Updates* ist eigentlich selbsterklärend. Dieser aktiviert einen Dialog, welcher den Anwender darüber informiert, dass nach Updates gesucht wird. Diesen Punkt sollten Sie immer dann ausführen, wenn es mit Eclipse selbst ein Problem gibt. Es wird nicht nur nach Updates gesucht, auch zusätzliche Pakete werden automatisch erkannt und heruntergeladen. Im Moment ist aber der Eintrag *Install New Software...* wichtiger, dieser wird nämlich im folgenden Abschnitt benötigt, um das ADT-Plugin zu installieren.

2.3 Download und Installation des ADT-Plugins

Hinweis
Die Installation des ADT-Plugins ist nur notwendig, wenn nicht das ADT Bundle verwendet wird.

Eclipse alleine ist schon eine prima IDE, aber leider für die Entwicklung von Android-Apps noch ungeeignet. Die fehlenden Bausteine müssen also in Form eines Plugins installiert werden. Dieses Plugin, ADT genannt, erhält man von der Android-Developer-Webseite unter folgender URL: *http://developer.android.com/sdk/eclipse-adt.html*. Allerdings muss man das Plugin nicht zwangsläufig herunterladen. Es gibt zwei Möglichkeiten der Installation. Die erste Variante ist, man lädt die Installationsdatei für das Plugin herunter und »sagt« Eclipse anschließend, wo die Datei liegt und dass sie installiert werden soll. Wählt man den zweiten Weg, so führt Eclipse den Download und die anschließende Installation selbstständig durch. Variante zwei funktioniert allerdings nur, wenn man nicht hinter einer Firewall sitzt oder evtl. benötigte Ports gesperrt sind.

Im Folgenden wird erst Variante zwei vorgestellt, da die meisten diesen Weg bevorzugen. Er ist einfach bequemer.

Nachdem Start der Eclipse-IDE ruft man im Menü *Help* den Punkt *Install New Software...* auf.

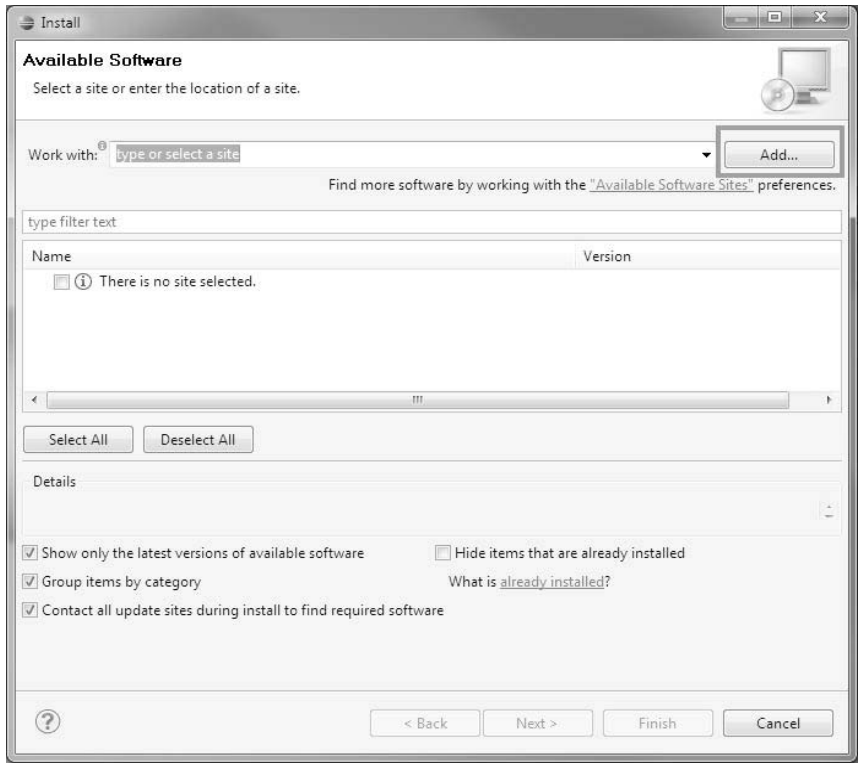


Bild 2.6: Installation des ADT-Plugins vorbereiten

In der rechten oberen Ecke befindet sich eine Schaltfläche mit dem Namen *Add* (siehe Bild 2.6), diese muss betätigt werden. Als Nächstes öffnet sich ein weiterer Dialog: *Add Repository*.

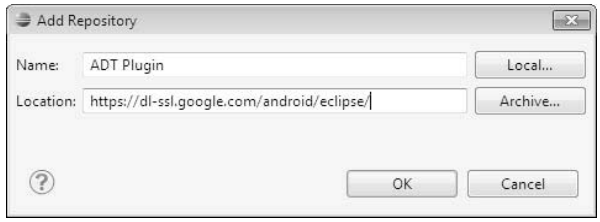


Bild 2.7:
Der *Add Repository*-Dialog

Im *Add Repository*-Dialog wird zuerst der Name (ADT-Plugin) des zu installierenden Plugins erfasst. Anschließend wird unter *Location* die Quelle der Installation eingetragen. Hier wird die folgende URL hinterlegt: *https://dl-ssl.google.com/android/eclipse*. Betätigt man anschließend den *OK*-Button, so erscheint ein *Install*-Dialog. Dort muss man nun das im Fenster angezeigte Paket *Developer Tools* selektieren und zuletzt die *Next*-Schaltfläche betätigen.

Bild 2.8: Installation des ADT-Plugins

Nun läuft die Installation des Plugins, dieser Vorgang kann ein wenig dauern. Zuletzt muss man noch den Lizenzbedingungen zustimmen und Eclipse neu starten.

Alternative Installation

Wenn man die Installation via HTTP nicht durchführen kann, z. B. wegen einer Firewall, dann muss man zunächst das komplette ADT-Plugin bzw. dessen Installationsdatei vollständig herunterladen. Das ADT-Setup-Paket ist ebenfalls unter *http://developer.android.com/sdk/eclipse-adt.html* verfügbar.

Zur Installation geht man eigentlich genauso wie bei der HTTP-Variante vor. Im *Add Repository*-Dialog geben Sie allerdings unter *Location* keine URL an, sondern betätigen stattdessen den *Archive*-Button. Über den anschließend angezeigten Datei-Öffnen-Dialog selektiert man nun das zuvor heruntergeladene ZIP-Archiv und betätigt den *Öffnen*-Button. Natürlich muss man auch in diesem Fall einen Namen für das Plugin vergeben. Der Rest der Installation unterscheidet sich nicht von der HTTP-Variante.

Gelegentlich können Sie über den Punkt *Check for Updates* prüfen, ob eine neue Version des ADT-Plugins vorliegt.

2.4 Neues Android-Projekt mit Eclipse

Bisher haben Sie sich fast ausschließlich theoretisch mit der Entwicklung für Android beschäftigt. Zeit für eine kleine Fingerübung. An dieser Stelle könnte eine Hallo-Welt-App nicht schaden. Erstellen Sie also zuerst ein neues virtuelles Android-Phone. Zur Arbeitsweise mit einem via USB angeschlossenen Android-Handy kommen wir etwas später. Das virtuelle Phone wird automatisch mit dem Projekt gestartet, das bedeutet, es muss nicht vorher manuell aktiviert werden.

Legen Sie also zunächst ein neues Projekt an. Im *File*-Menü wird *New > Project...* gewählt. Als Nächstes öffnet sich der Dialog *New Project*, dieser enthält die möglichen Projekttypen. Hier wählen Sie unterhalb von *Android* den Eintrag *Android Application Project* aus und betätigen die *Next*-Schaltfläche. Anschließend wird der Dialog *New Android Application* geöffnet (vgl. Bild 2.9). In diesem Dialog wird der Rahmen für ein neues Projekt gesteckt und z. B. der Name der App sowie die unterstützten Android-Versionen festgelegt. Die komplette Anlage des Projekts besteht aus mehreren Schritten, welche jeweils in einem eigenen Dialog angezeigt werden. Im ersten Feld wird der

Projektname eingegeben (Feld *Application Name*), im Beispiel AppBsp. Der Application Name wird im Google Play Store und in der Anwendungsliste des Android-Geräts angezeigt. Im folgenden Feld *Project Name* wird der Name des Projekts eingegeben. Der Projektname wird nur von Eclipse verwendet und muss im Workspace von Eclipse eindeutig sein. Per Voreinstellung wird der *Application Name* auch als *Project Name* übernommen. Man sollte es dabei belassen und nur in begründeten Ausnahmefällen davon abweichen. Im letzten Feld wird nun der *Package Name* festgelegt.

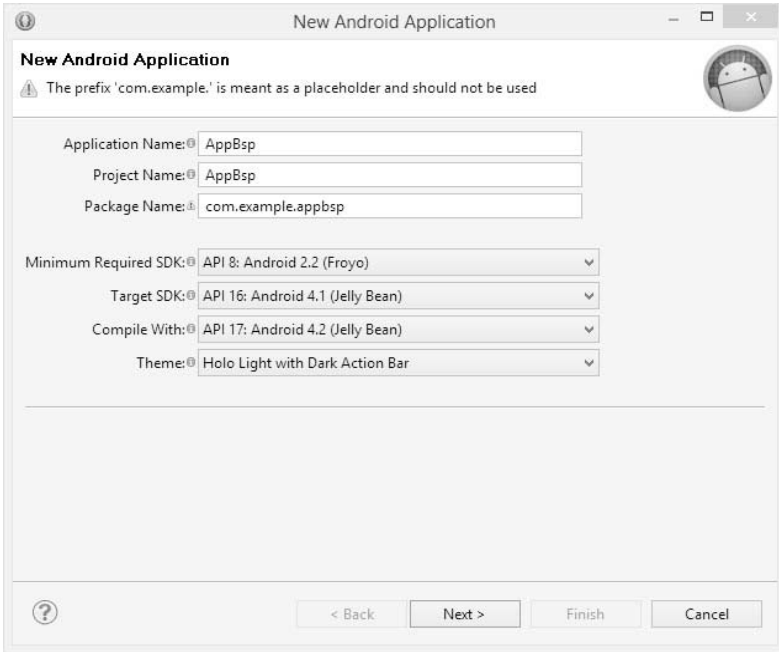


Bild 2.9: Anlage eines neuen Android-Projekts

Mit einem *Package* wird der Programmcode, den Sie schreiben, in logische Bereiche aufgeteilt. Programmteile, die zusammengehören, sollten natürlich auch immer im selben *Package* eingefügt werden. Stellen Sie sich ein *Package* am besten wie ein Verzeichnis auf der Festplatte vor, das ggf. weitere Verzeichnisse enthält. Der Verzeichnispfad *Dokumente\Briefe\Anschreiben* würde in Java wie folgt geschrieben werden: *dokumente.briefe.anschreiben*. Ein Brief, der im Ordner *Anschreiben* liegt, würde somit zum Package *anschreiben* gehören. Android-Packages strukturiert man eigentlich genauso, nur dass eine andere Schreibweise verwendet wird. Oft findet man auch eine Package-Bezeichnung, welche den Firmennamen enthält, z. B. *de.firmenname.projektname*.

Im Feld *Package Name* wird auch an dieser Stelle eine automatische Konfiguration vorgenommen und *com.example* plus *Application Name* / *Project Name* eingetragen. Hier sollte man natürlich eine Änderung vornehmen (für das Hallo-Welt-Beispiel können Sie es aber auch bei der Voreinstellung belassen). Wenn Ihnen kein Packagename einfällt, dann verwenden Sie einfach *de* plus *Application Name*. Das *de* steht in diesem Fall für

Deutschland. Der *Package Name* könnte also beispielsweise folgendermaßen aussehen: *de.appbsp*.

Im nächsten Arbeitsschritt muss festgelegt werden, welche Version des Android-SDKs wofür verwendet werden soll. Mit der ersten Option *Minimum Required SDK* wird festgelegt, welche Version von Android auf einem Smartphone zumindest installiert sein muss, damit die App auf dem Gerät ausgeführt werden kann.

Bild 2.10: Auswahl des Android-SDKs

Die Auswahl des APIs ist natürlich auch abhängig davon, welche SDKs Sie auf Ihrem System installiert haben. Im Beispiel wurde das SDK 2.2 gewählt. Das ist aber nicht zwingend erforderlich. Ein Hallo-Welt-Projekt läuft genauso gut unter Version 1.6 des API. Wichtig ist nur, dass Sie auch einen passenden Emulator angelegt haben. Wenn Sie sich also für das 1.6-API entscheiden, dann benötigen Sie auch einen Emulator, der Android 1.6 unterstützt. Die höheren SDKs benötigt man eigentlich nur, wenn man Funktionen verwendet, welche in den niedrigen Versionen nicht verfügbar sind. Ein älteres SDK bedeutet, dass die fertig gestellte App auch auf einer größeren Anzahl an Geräten läuft.

Google unterstützt Sie bei der Auswahl eines geeigneten SDKs. Unter folgendem Link *http://developer.android.com/resources/dashboard/platform-versions.html* gibt es eine Übersicht, die 14-täglich aktualisiert wird. Auf diese Seite sollten Sie vor Beginn der Entwicklung immer einen Blick werfen, um zu entscheiden, ob es sich »lohnt«, eine bestimmte Funktion zu verwenden. Auch ein weiteres Detail kann wichtig sein. Je höher das verwendete SDK ist, desto mehr Ressourcen benötigt meist auch der passende Emulator. Das heißt, wenn Sie einen langsamen Rechner haben, kann es durchaus angenehmer sein, wenn Sie zu Beginn eine ältere SDK-Version verwenden (sofern die geplanten Funktionen vom SDK unterstützt werden).

Die nächste Option ist *Target SDK*. Mittels dieser Option gibt man an, bis zu welcher Android-Version (meist die neueste) man eine App getestet hat bzw. auf welcher die App ausgeführt werden kann. Keine Sorge, diese Option schließt natürlich nicht aus, dass die App auch auf zukünftigen Versionen von Android ausgeführt werden kann. Es folgt die Option *Compile With*. An dieser Stelle legt man fest, gegen welche API-Version

das Projekt kompiliert wird. In der Regel ist dies die aktuellste Version von Android bzw. des zugehörigen SDKs. Eine letzte Auswahl muss man in diesem Arbeitsschritt noch treffen. An dieser Stelle kann man sich nämlich auch entscheiden, welches Android-Theme verwendet werden soll. Zur Auswahl stehen: None, Holo Dark, Holo Light sowie Holo Light with Dark Action Bar. Die Auswahl lässt sich später natürlich noch ändern. Aber Achtung, das Holo Theme gibt es erst ab API Version 11, also Android 3.0. Entscheidet man sich für die Verwendung von Holo, so muss für ältere Versionen von Android eine alternative Konfiguration in der Datei *styles.xml* im Ordner *values* eines Projekts hinterlegt werden. Glücklicherweise wird bei der Anlage eines Projekts und entsprechender Auswahl automatisch eine entsprechende Datei generiert. Auch hier gilt für das Hallo-Welt-Beispiel: Belassen Sie es bei den voreingestellten Einträgen.

Nach der letzten Auswahl muss die *Next*-Schaltfläche betätigt werden. Anschließend gelangen Sie zur Konfiguration des Projekts.

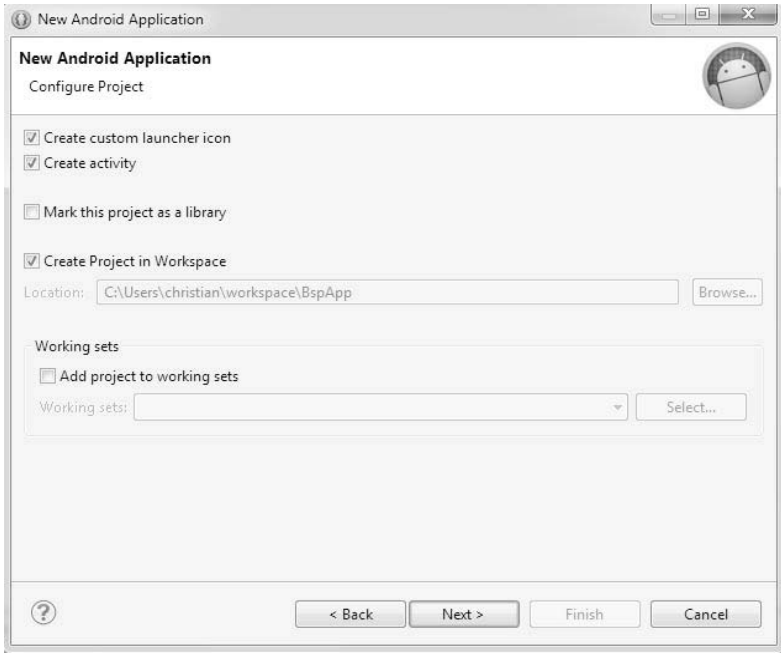


Bild 2.11: Konfiguration eines Projekts

Mit der Option *Create custom launcher icon* legen Sie fest, ob im nächsten Dialog (*Configure Launcher Icon*) das App-Icon individuell konfiguriert werden soll. Wird die Auswahl entfernt, so wird der eigentlich folgende Arbeitsschritt übersprungen und der kleine Droide wird als Symbol für Ihre App verwendet. Die direkt folgende Option *Create activity* wird verwendet, um festzulegen, ob automatisch eine Activity angelegt werden soll.

Hinweis
Eine Activity ist eine Klasse, also ein Rahmen für Programmcode. Oft wird sie in Verbindung mit einem (XML-)Formular verwendet und enthält dann Code, um auf Benutzeraktionen (z. B. die Betätigung einer Schaltfläche) zu reagieren. In Kapitel 5 finden Sie mehr Informationen zu diesem Thema.

Wenn man kein eigenständiges Projekt erzeugen möchte, sondern nur eine Bibliothek, welche auch in anderen Projekten (Apps) verwendet werden soll, dann muss die Option *Mark this project as a library* aktiviert werden.

Soll das Projekt nicht im Standard-Workspace gespeichert werden, sondern in einem anderen Ordner, so muss die Option *Create Project in Workspace* deaktiviert und ein anderes Zielverzeichnis ausgewählt werden. Mit dem letzten Punkt kann man Projekte zu *Working Sets* hinzufügen. Hat man viele Projekte mit Eclipse erstellt, so wird es irgendwann einmal unübersichtlich. Zu diesem Zeitpunkt kann es sinnvoll sein, Projekte zu sogenannten *Working Sets* zusammenzufassen. In Eclipse werden dann nur noch die Projekte eines ausgewählten *Working Sets* direkt angezeigt. Natürlich kann man zwischen den Working Sets wechseln.

Hinweis
Auch auf die Gefahr hin, dass ich mich wiederhole – für das Hallo-Welt-Beispiel ist es am besten, die voreingestellte Auswahl zu übernehmen.

Sofern die Option *Create custom launcher icon* aktiviert wurde, wird im folgenden Arbeitsschritt das Launcher Icon der App konfiguriert.

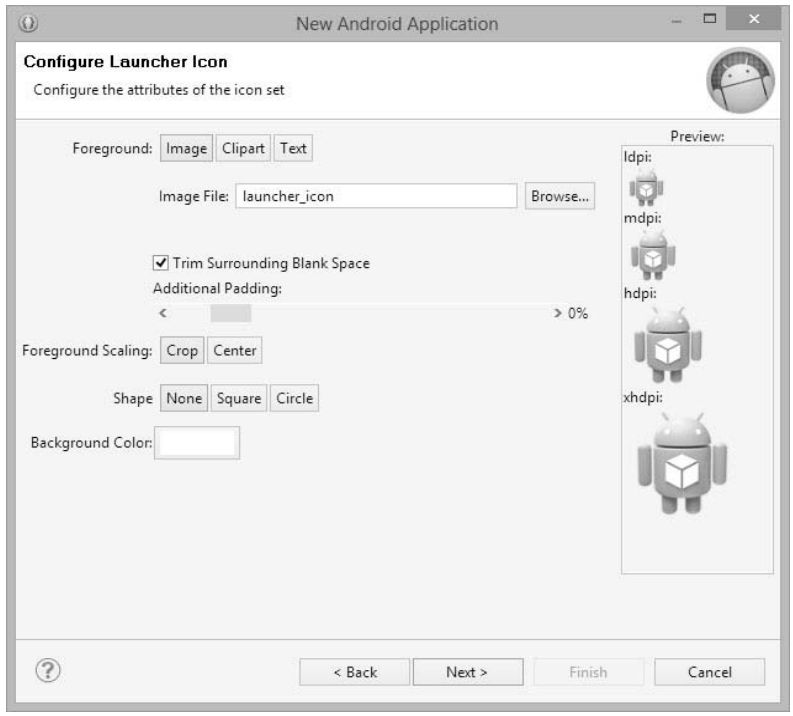


Bild 2.12: Konfiguration des App-Launcher-Icons

An dieser Stelle hat man viele Möglichkeiten, Einfluss auf die Darstellung des Icons zu nehmen. Zuerst muss man sich entscheiden, ob man ein eigenes Image für das Icon verwendet oder aber auf eine Grafik aus der Clipart-Bibliothek zurückgreift. Auch die Verwendung von Text als »Icon« ist möglich. Gesteuert wird dies durch die drei Schaltflächen *Image*, *Clipart* und *Text*. Ein Mausklick öffnet jeweils ein zur Konfiguration passendes Menü. Entscheidet man sich für ein Image als Launcher Icon, so muss man an dieser Stelle berücksichtigen, dass die Grafik in vierfacher Ausfertigung benötigt wird. Jeweils in den passenden Größen für ldpi (120 dpi) 36 x 36 Pixel, mdpi (160 dpi) 48 x 48 Pixel, hdpi (240 dpi) 72 x 72 Pixel und xhdpi (320 dpi) 96 x 96 Pixel. Als Alternative, wenn man keine eigene Grafik hat, kann man den kleinen Droiden zum Ausprobieren verwenden. Zur Bearbeitung des Launcher Icons stehen dann noch weitere Funktionen bereit. Mittels der Optionen *Trim Surrounding Blank Space* und *Additional Padding* kann die Grafik so angepasst werden, dass die zur Verfügung stehende Fläche komplett für die Grafik innerhalb des Icons verwendet wird. Zusätzlich besteht die Möglichkeit, den Hintergrund der Grafik etwas anzupassen. Es kann die Farbe (*Background Color*) sowie die Form (Keine, Quadrat, Kreis) gewählt werden.

Nur wenn man im Arbeitsschritt *Configure Project* die Option *Create activity* gewählt hat, kann man im vierten Arbeitsschritt (*Create Activity*) eine Vorlage auswählen, nach welcher die neu zu erstellende Activity gestaltet wird.

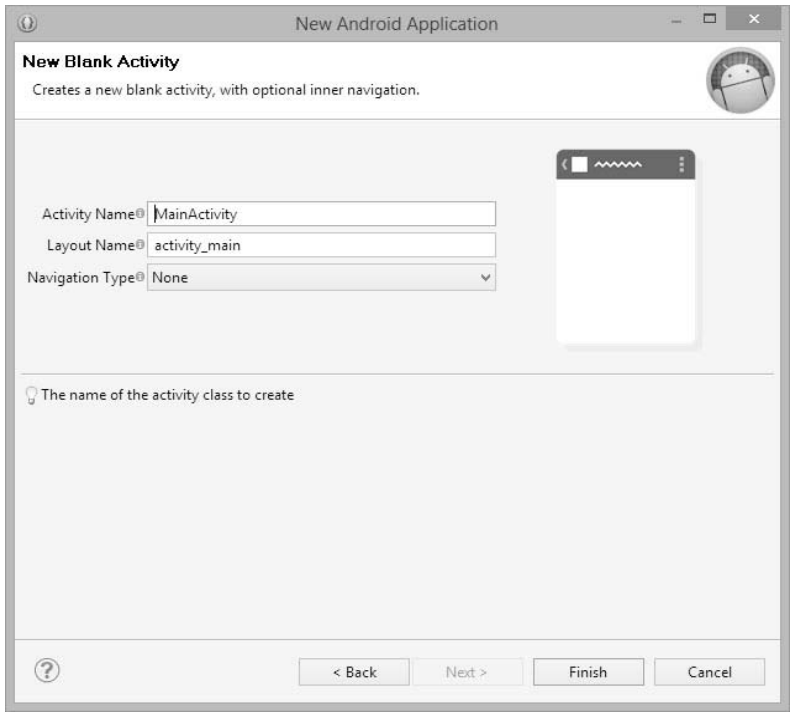


Bild 2.13: Konfiguration der Main-Activity

Zu Beginn des Dialogs kann man sich immer noch gegen eine Anlage entscheiden und diesen Schritt überspringen. Ansonsten muss man nun wählen: Soll eine leere Activity (*BlankActivity*) angelegt werden oder eine der vier weiteren Optionen? Wählt man *FullscreenActivity*, so wird das Layout so gestaltet, dass die Status Bar sowie eine evtl. vorhandene Navigation oder System Bar automatisch ein- und ausgeblendet werden. Die Vorlage *LoginActivity* erzeugt automatisch ein Layout, welches über Felder zur Eingabe von Benutzername und Passwort sowie einer Schaltfläche zum Bestätigen der Eingabe verfügt. Mit der Auswahl von *MasterDetailFlow* wird ein Layout erzeugt, welches im linken Bereich über ein Auswahlmenü und im rechten Bereich über eine Detaildarstellung verfügt. Die Verwendung dieser Vorlage mündet in der Anlage von mehreren Activities/Layoutdateien. Zur Anwendung kommt hier die Fragment-Technik, welche in Kapitel 15.3 beschrieben wird. Die Anwendung dieser Vorlage ist nur dann möglich, wenn zuvor mindestens das API 11 (Android 3.0) als Basis für die App gewählt wurde. Die letzte Vorlage erzeugt ein sogenanntes *SettingsActivity*. Es handelt sich hierbei um die Möglichkeit, einen Bildschirm für die Konfiguration von Anwendungseinstellungen einer App zu erzeugen. Kapitel 17 liefert Detailinformationen zu diesem Thema. Mit einem weiteren Mausklick auf die Next-Schaltfläche gelangt man zum letzten Arbeitsschritt.

Was das Hallo-Welt-Beispiel betrifft, so wird die Vorlage *BlankActivity* (also die Voreinstellung ☺) verwendet.

Hinweis
Der neue Dialog mit den Activity-Vorlagen ist eine praktische Sache. Wird der Entwickler doch an dieser Stelle entlastet und muss nicht für wiederkehrende Aufgaben eigene (Copy & Paste)-Vorlagen erstellen. Das ADT-Team sieht das genauso. Aus diesem Grund kann der *Create Activity*-Dialog jederzeit aufgerufen werden. Einfach das Projekt markieren und über das Kontextmenü den Punkt *New* auswählen. Im anschließend angezeigten Untermenü dann den Menüpunkt *Other* auswählen. Im dann angezeigten Dialog kann man im Ordner *Android* den Punkt *Android Activity* auswählen, welcher den *Create Activity*-Dialog öffnet.

Der letzte Arbeitsschritt beinhaltet lediglich die Vergabe von Bezeichnungen für die neue Activity sowie die zugehörige Layoutdatei. Nur wenn Sie zuvor die Vorlage *BlankActivity* gewählt haben, haben Sie hier zusätzlich die Möglichkeit, einen *Navigation Type* festzulegen. Über das Auswahlfeld *Navigation Type* wird festgelegt, ob die Activity mit zusätzlichen *Tabs* (also Registerkarten) ausgestattet werden soll. Neben der Auswahl *Tabs* gibt es zusätzlich noch den Punkt *Tabs + Swipe*, welcher bewirkt, dass die einzelnen Tabs mit einer Swipe-Funktion versehen werden (auch hier gilt: erst ab API Level 11 möglich). Mit der Auswahl von *Swipe Views + Title Strip* tauscht man die Tabs gegen Views, welche via Swipe-Geste gewechselt werden. Zuletzt gibt es noch die Möglichkeit, mit der Auswahl von *Dropdown* ein Pulldown-Menü in die Titelleiste der App zur Navigation zu integrieren.

Betätigt man jetzt die Schaltfläche *Finish*, so wird die Android-App nach den eingegebenen Parametern automatisch erstellt. Anschließend wird das Projekt im Package Explorer (am linken Rand) angezeigt.

Für das Hallo-Welt-Beispiel gilt auch an dieser Stelle: Keine Änderung! Einfach den Finish-Button betätigen und: *Sie haben fertig*!

Hinweis
Die neueste Version der Werkzeuge für Android-Entwickler legt vor allem Wert auf leichtere Zugänglichkeit. Gerade Einsteiger profitieren von den neuen Vorlagen für Activities, welche sich auch in laufenden Projekten nutzen lassen.

2.4.1 Projektübersicht mit Package Explorer & Start einer App

In der Standardeinstellung direkt nach der Installation von Eclipse befindet sich der Package Explorer im linken Bereich der Entwicklungsumgebung. Nach der Anlage des Projekts wird dieses im Package Explorer angezeigt. Es ist dort allerdings nur mit seinem Namen zu sehen. Bestandteile sind nicht zu erkennen. Ein Doppelklick auf den Projektnamen öffnet den Projektbaum und die diversen Elemente eines Projekts sind zu sehen. Jedes Mal wenn ein neues Projekt angelegt wird, ist dieses nach Anlage im Package Explorer zu finden. Es ist möglich, sich gezielt einzelne Bestandteile des Projekts anzusehen. Doppelklickt man beispielsweise auf den Ordner mit dem Namen *res*, so werden weitere untergeordnete Elemente angezeigt, welche ebenfalls zum Projekt gehören. Ein Doppelklick auf den Ordner *layout* bringt ein weiteres Element mit dem Namen *main.xml* zum Vorschein. Führt man jetzt einen Doppelklick auf dem Element *main.xml* aus, so wird im rechten Bereich der IDE ein neues Fenster geöffnet, in dem ein Android-Screen zu sehen ist. Es sind noch weitere Elemente sichtbar. So gibt es im linken Teil einen Bereich, welcher mit dem Namen *Palette* bezeichnet ist. Aus der *Palette* können Elemente in den Screen via Drag & Drop befördert werden. Die *Palette* enthält sogenannte Controls. Das sind beispielsweise ein Button oder auch ein Spinner. Elemente also, mit denen sich eine Oberfläche für eine Android-App »bauen« lässt.

Hinweis zum Project Explorer
Wenn Sie ein wenig in Eclipse herumstöbern, so werden Sie vermutlich früher oder später auch auf den *Project Explorer* stoßen. Vergleicht man Package und Project Explorer, so wird man auf den ersten Blick keinen Unterschied feststellen können. Der Package Explorer ist allerdings eine Ansicht, die speziell für Java-Projekte gedacht ist (daher auch der Name). Der Project Explorer hingegen kann auch für Projekte in anderen Sprachen (z. B. PHP, C, C++ usw.) verwendet werden.

App im Emulator starten

Der Package Explorer hilft dem Entwickler also, durch ein Projekt zu navigieren und einzelne Elemente aufzurufen, um diese zu bearbeiten. Wie kann man jetzt das soeben erzeugte Projekt starten?

Markus Spiering / Sven Haiges / René Scholze

HTML5-Apps für iPhone und Android

HTML5, CSS3 und jQuery Mobile:
Design, Programmierung und Veröffentlichung
plattformübergreifender Apps

Inhaltsverzeichnis

Einführung

Mit der Einführung des iPhones im Jahr 2007 hat sich die mobile Welt komplett geändert: Internetdienste werden von einer großen Masse wie selbstverständlich genutzt, und der mobile Markt hat sich, anstatt sich auf einer oder einigen wenigen Plattformen zu konsolidieren, in Höchstgeschwindigkeit weiter fragmentiert. Apple hat mit dem iPhone und iPhone OS ein Erfolgsmodell geschaffen und ein unvergleichliches Wettrennen der mobilen Plattformen ausgelöst.

Einige Jahre nach dem ersten iPhone haben wir folgende Situation: Die Verbindung von iPhone und iTunes hatte ein Verkaufsmodell für mobile Anwendungen geschaffen, das Hunderttausende von Anwendungen für die iOS-Plattform hervorgebracht hat und Entwickler begeistert. Microsoft, zwischenzeitlich schon in der mobilen Bedeutungslosigkeit verschwunden, hat nun mit Windows Phone 8 ein interessantes Betriebssystem auf den Markt gebracht, das sich bald auf allen Nokia-Geräten finden wird. Handy-Weltmarktführer Nokia hatte sich jahrelang auf seine Symbian-Plattform verlassen, die sich auch heute noch weltweit auf den meisten Handys befindet, jedoch nicht mehr mit den modernen mobilen Plattformen mithalten kann. Bevor sich Nokia entschied, mit Microsoft und Windows Phone 8 eine Ehe einzugehen, plante der Konzern, die mit Intel zusammen entwickelte MeeGo-Platform zu verwenden.

BlackBerry hat keine andere Plattform lizenziert, sondern setzt weiterhin auf sein eigenes System. Palm, mittlerweile von HP gekauft und schon vor einigen Jahren tot geglaubt, hat zahlreiche Apple-Designer und -Techniker abgeworden und mit Web OS ein unglaublich performantes und zukunftsweisendes mobiles Betriebssystem auf den Markt gebracht. Die Web-OS-Plattform wird derzeit auf Mobiltelefonen sowie Tablet-Computern von HP selbst benutzt, und es bleibt abzuwarten, wie sich diese Plattform entwickelt. Trotz dieser Plattformvielfalt gab Google im November 2007 bekannt, gemeinsam mit 33 Mitgliedern, der sogenannten Open Handset Alliance, ein weiteres mobiles Betriebssystem namens Android zu entwickeln. Ohne Zweifel ist Android das derzeit am schnellsten wachsende Betriebssystem und weltweit auf den meisten Telefonen zu finden.

Ohne die hausgemachten Systeme von Samsung, LG und anderen Herstellern, besonders aus dem asiatischen Raum, zu erwähnen, kommen wir mit Apple iOS, Android, Windows Phone 8, BlackBerry, Symbian, MeeGo, Web OS und mobilen Linux-Versionen auf acht verschiedene nennenswerte Handyplattformen.

Zur gleichen Zeit hat sich mit dem iPhone die Handynutzung verändert: Während man Telefone größtenteils zum Telefonieren, Verfassen von Textnachrichten oder zum Checken von E-Mails verwendete, wurde das iPhone als erstes Telefon serienmäßig mit einem Browser ausgeliefert, der einem Browser auf dem PC in nichts nachsteht. Mit dem App-Modell, aber auch der iTunes-Anbindung wurde der eingestaubte Markt von mobilen Anwendungen auf den Kopf gestellt. Ein Verkaufsargument für Handys ist heute nicht mehr nur der bessere Bildschirm oder die bessere Kamera, sondern auch die

Art von Diensten, die ich auf dem Gerät bzw. der Plattform nutzen kann. Kann ich das Telefon mit iTunes verbinden und meine Musik synchronisieren? Kann ich meinen Facebook-Status aktualisieren, meine Fotos direkt auf Flickr laden und mit meinen Freunden und Bekannten direkt vom Telefon aus teilen? Wie viele Programme und auch Spiele gibt es eigentlich für mein Telefon?

Und hier beginnt genau das Problem für Handyhersteller, für Anwender und auch für Anbieter von Programmen und Diensten. Alle Plattformen außer derzeit iOS von Apple und Android von Google haben Probleme, im großen Rahmen Entwickler für ihre Plattform zu finden und zu begeistern. Wurden mobile Softwareentwickler vor ein paar Jahren noch kaum beachtet, so sind sie jetzt heiß umworbene Dienstleister. Alle Plattformanbieter betreiben einen riesigen Aufwand, um Entwickler davon zu überzeugen, mit ihrem System zu arbeiten und Programme und Dienste zu schreiben.

Gleichzeitig müssen sich Anbieter von Diensten und Programmierer die Frage stellen, auf welche Plattformen sie sich konzentrieren sollten. Alle Plattformen sind grundverschieden, und Code, der für das iPhone geschrieben wird, kann nicht einfach für Android, BlackBerry oder für Windows Phone 8 wiederverwendet werden. Die Entwicklung von mobilen Diensten ist für Firmen notwendig, auch wenn der mobile Markt viel verspricht, aber bislang nur wenig abwirft. Gleichzeitig ist die Entwicklung von mobilen Programmen sehr teuer, sollte von mobilen Softwarespezialisten vorgenommen werden und verursacht allein durch das Testen auf verschiedenen Telefonen hohe Kosten. Keine Firma wird so einfach bereit sein, gleichzeitig in alle acht Plattformen zu investieren, sondern sich höchstens zwei oder maximal drei unterschiedliche Versionen leisten. Doch es gibt Hoffnung: Neben großen, hochauflösenden Bildschirmen und tollen Kameras werden alle Handys in naher Zukunft eine Gemeinsamkeit aufweisen: unwahrscheinlich gute Webbrowser, die sich nach den neuesten Standards richten. Der neueste Webstandard, der mehr und mehr von den gängigen Webbrowsern unterstützt wird, ist HTML5.

Glücklicherweise wird sich HTML5 von der HTML4-Spezifikation insbesondere dadurch unterscheiden, dass sich mit der neuen Version Webdienste erstellen lassen, die nicht nur wie richtige Programme aussehen, sondern sich auch wie Programme verhalten können. Auf Inhalte einer Webseite kann auch dann zugegriffen werden, wenn keine Internetverbindung vorhanden ist, auf das Dateisystem kann vom Internetbrowser aus zugegriffen werden, und auch die aktuelle Position des Benutzers ist ab sofort für eine Internetseite kein Geheimnis mehr. Alle diese Funktionen konnten früher nur von richtigen Programmen, aber nie von Internetseiten verwendet werden.

Aufgrund der genannten Probleme und der starken Fragmentierung des mobilen Markts, jedoch auch wegen der herausragenden Möglichkeiten, die moderne mobile Webbrowser und der HTML5-Standard bieten, muss man annehmen, dass HTML5 insbesondere im mobilen Bereich einen enormen Einfluss nehmen wird.

Für Dienstanbieter ist das ein Glücksfall: Sie können auf vorhandene Webprogrammierer zurückgreifen, mit einer mobilen Webanwendung in HTML5 sicherstellen, dass sie einen sehr guten Dienst auf allen mobilen Plattformen anbieten, und sie können sich immer noch entscheiden, eine native Anwendung für beispielsweise den iPhone App Store oder den Google Play Store programmieren zu lassen. Das ist auch gar nicht schwer: Mittler-

weile gibt es mit dem PhoneGap (*www.phonegap.com*) und dem Appcelerator Titanium Mobile (*www.appcelerator.com*) Technologien, die aus HTML5-Webanwendungen native Anwendungen für die gängigsten mobilen Plattformen bauen.

Insbesondere mit PhoneGap lassen sich im Handumdrehen richtige Anwendungen erstellen, die nicht nur in den App-Stores verkauft werden, sondern sich auch mit Funktionen bereichern lassen, die im Browser gar nicht zur Verfügung stehen. Die Bewegungssensoren, die Kamera, der Vibrationsalarm, der Kompass, das Adressbuch und vieles mehr lassen sich über ein paar simple JavaScript-APIs direkt vom PhoneGap-, HTML-, CSS- und JavaScript-Code aus steuern.

All diese Technologien und Ansätze möchten wir Ihnen auf den nächsten gut 300 Seiten vorstellen. Am Ende des Buchs haben Sie nicht nur ein umfassendes Wissen über HTML5, sondern auch das Zeug, Ihre erste iPhone-App oder Android-Anwendung in iTunes bzw. dem Google Play Store zu veröffentlichen.

Worum geht es?

Dieses Buch ist kein Programmierbuch im klassischen Sinne. Es gibt Ihnen einen Einblick in Designkonzepte von mobilen Webdiensten und hilft Ihnen, erste Webanwendungen mittels HTML, CSS und JavaScript zu erstellen. Diese Webanwendungen werden bereits ein richtiges »Look-and-feel«, wie man es auf Smartphones kennt, besitzen. Ein großer Teil des Buchs beschäftigt sich mit den neuen Funktionen und Möglichkeiten von HTML5 und zeigt diese im mobilen Kontext eindrucksvoll und verständlich anhand vieler Beispiele. Der letzte Teil des Buchs beschreibt Schritt für Schritt, wie Sie eine HTML5-Webanwendung in eine echte iPhone- oder Android-Anwendung umwandeln und dabei Telefonhardwarefunktionen wie die Kamera oder den Bewegungssensor eines iPhones oder Android-Smartphones ausnutzen. Abschließend erfahren Sie, wie Sie Ihre frisch gebaute Anwendung in den Apple Store und in den Google Play Store bekommen.

Die Twitter-Updates zum Buch
Folgen Sie *@html5togo* auf Twitter – *http://twitter.com/html5togo* –, um direkt von den Autoren des Buchs neueste Updates und interessante Artikel zum Thema HTML5 und Mobile zu erhalten und mit ihnen in Kontakt zu treten.

HTML5 – die Spezifikation und der aktuelle Stand

Streng genommen ist HTML5, das sich noch in der Entwicklung befindet, eine neue Version des HTML-Standards. Wie auch HTML4.1 und XHTML 1.1 ist HTML5 eine standardisierte Markup-Sprache, die dazu dient, Webinhalte zu strukturieren und zu präsentieren. Rein technisch gesehen, ist HTML5 aber eben HTML und nicht JavaScript und auch nicht CSS.

Doch die Masse versteht unter HTML5, speziell unter dem Begriff »HTML5 Web Apps«, etwas anderes. HTML5 ist zum Schlagwort einer (begrüßenswerten) Entwicklung geworden und beschreibt Anwendungen, die mit Webstandards, also HTML, CSS und JavaScript, geschrieben werden und in den Browsern auf unterschiedlichen Plattformen laufen. HTML5 steht für Webseiten, die sich wie Anwendungen anfühlen, wie Anwendungen gestartet werden können und weit in den Bereich nativer Anwendungen vordringen. Wir versuchen in diesem Buch, die richtige Balance zu finden zwischen neuen Funktionen, die sich wirklich auf HTML5 als Markup-Sprache beziehen, und den Möglichkeiten, die durch eine Vermischung von HTML, JavaScript und CSS entstehen und eben auch als »HTML5« bezeichnet werden.

Auch wenn Sie überall schon von HTML5 hören, heißt das noch nicht, dass die neue Version des HTML-Standards bereits fertig ist. Im Gegenteil: HTML5 ist derzeit noch in der Phase des »Working Draft« – also des Stadiums, in dem immer noch aktiv an der Definition der Sprache gearbeitet wird. Allerdings werden nicht alle Komponenten, also nicht alle Tags gleichzeitig bearbeitet, und bestimmte Teile der Sprache sind bereits weiter vorangeschritten als andere. Komponenten wie beispielsweise das `canvas`-Tag sind sehr stabil und nahezu »fertig«, während andere Teile der Sprache noch gar nicht definiert wurden.

Der Entwicklungszyklus einer neuen HTML-Definition bewegt sich im Bereich von 20 Jahren. Mit der HTML5-Spezifizierung wurde 2004 unter dem Namen »Web Applications 1.0« begonnen, und es wird derzeit erwartet, dass die Sprache im Jahr 2022 die offizielle W3C-Empfehlung erhalten wird. Zum Vergleich: Die derzeit gültige HTML5-Spezifikation wurde Mitte der 90er-Jahre begonnen.

Nun sollten Sie auf keinen Fall bis 2022 warten, sondern die Möglichkeiten von HTML5 und den oft in einem Atemzug genannten Technologien JavaScript und CSS schon heute nutzen. HTML5 ist bereits da, wird aber in der Zukunft noch wesentlich umfangreicher und von mehr Plattformen und Browsern unterstützt werden, als es heute der Fall ist. Die hier im Buch vorgestellten HTML5-Methoden sind schon größtenteils von den führenden mobilen Plattformen implementiert, ganz vorn sind natürlich iOS und Android, mit dabei sind auch BlackBerry OS 6 und Windows Phone 8.

Lesezeichen
http://dev.w3.org/html5/spec/
Offizielle HTML5-Spezifikation: In diesem Buch wird die offizielle HTML5-Spezifikation immer in einem mobilen Umfeld betrachtet. Interessieren Sie sich für die komplette Spezifikation, finden Sie sie hier.

Webanwendung vs. native Anwendung

Bereits weiter oben wurde häufig von einer Webanwendung und einer »richtigen« nativen Anwendung gesprochen. Auch wenn HTML5 die Grenzen und Möglichkeiten

zwischen einem richtig programmierten Programm und einer Webseite verschwimmen lässt, so gibt es doch einige nennenswerte Unterschiede, die zu bedenken sind.

Native Anwendung

Unter einer nativen Anwendung versteht man ein Programm, das nicht im Browser des Telefons (oder eines Computers) läuft, sondern als eigenständiges Programm auf das Gerät geladen wurde und ein richtiges Computerprogramm darstellt. Native Anwendungen haben in der Regel einen besseren Zugriff auf Systemfunktionen, wie beispielsweise die Kamera eines Handys oder Bewegungssensoren, können Rechenoperationen viel schneller ausführen und die Grafikmöglichkeiten eines Systems wesentlich besser ausnutzen. Sämtliche Spiele, die auf schnelle Grafikroutinen angewiesen sind, werden als native Anwendungen erstellt.

Diese Anwendungen haben gegenüber Webanwendungen nicht nur auf der technischen Seite Vorzüge, sondern bieten einen riesigen Vorteil: Man kann sie in die App-Stores der verschiedenen Plattformanbieter einstellen. Distribution ist das Zauberwort!

Eine native Anwendung kann beispielsweise im iPhone App Store eingestellt, beworben und vertrieben werden. Benutzer müssen nicht umständlich im Internet nach ihrer mobilen Webseite suchen, sondern finden ein fertiges Programm zum Installieren in iTunes oder im App Store des iPhones. Die Wahrscheinlichkeit, dass ein Kunde Ihren Dienst in einem App Store entdeckt, ist wesentlich höher, als wenn er bei Google oder Yahoo! danach sucht. Ein zusätzlicher Bonus: Nach dem Hinzufügen ist das Programm direkt auf dem Home-Bildschirm des iPhones oder Android-Handys verewigt.

Dies alles klingt gut, und Sie werden sich jetzt vielleicht fragen, warum Sie sich kein Buch zur iPhone-Programmierung mit Objective-C gekauft haben, doch leider ist nicht alles so einfach:

- Der Aufwand, eine native Anwendung zu erstellen, ist pro Plattform immens hoch. Zwar kann es Ihnen gelingen, auf einer Plattform eine hohe Reichweite Ihrer Anwendung und Ihres Angebots zu erreichen, allerdings bedienen Sie nur eine von vielen Plattformen.
- Bevor Ihre Anwendung in einem der bekannten App-Stores erscheint, muss sie einen Genehmigungsprozess bestehen. Dies gilt nicht nur, wenn Ihre Anwendung erstmalig eingestellt wird, sondern der Prozess muss bei jeder Aktualisierung wiederholt werden.

Der Genehmigungsprozess, insbesondere um in den iPhone App Store zu kommen, wurde in der Presse schon oft heiß diskutiert. Selbst Internetgrößen wie Google wurde der Eintritt in den Store beispielsweise mit der Google-Voice-Anwendung verwehrt. Interessanterweise hat sich Google damit beholfen, eine Webanwendung mit HTML5 – *www.youtube.com/watch?v=neiOa38DuqI* – zu erstellen, die fast identische Funktionen wie die eigentliche native Anwendung auf das iPhone bringt.

Warum eine App bei Apple abgelehnt wurde, wird in der Regel auch nicht sehr detailliert begründet. Wie bereits gesagt, eine Überprüfung ist sogar bei jeder neuen Pro-

grammversion erforderlich. Wenn Ihre Anwendung beispielsweise einen Fehler enthält, kann dieser nicht einfach wie bei einer Webseite sofort behoben werden, stattdessen müssen Sie ein neues Anwendungspaket bauen, dieses Paket auf verschiedenen Telefonen testen, es an Apple oder einen anderen Betreiber des App Store übermitteln, auf das Testergebnis warten, gegebenenfalls nachbessern und dann hoffen, dass die Anwendung zum Verkauf bzw. Download freigegeben wird.

Eine weitere Einschränkung, insbesondere wenn Sie eine native iPhone-Anwendung erstellen möchten: Sie müssen einen Apple-Rechner besitzen, Objective-C lernen und Apple für die Aufnahme als Entwickler bezahlen. Bei der Entwicklung einer App für Windows Phone 8 ist es genau andersherum: Ohne PC mit Windows geht ebenfalls nichts.

Webanwendung

Im Gegensatz zu einer nativen Anwendung braucht eine Webanwendung ein anderes Programm zur Ausführung: den Browser. Daher ist die Webanwendung auch nur so gut wie der Browser, in dem sie läuft. Funktionen, die der Browser nicht weitergeben kann, beispielsweise den Zugriff auf die Kamera eines Handys, kann demzufolge eine reine Webanwendung auch (noch) nicht nutzen.

Den Vorteil, dass eine Webanwendung über einen App-Store verbreitet und sichtbar gemacht werden kann, besitzt leider auch die native Anwendung. Insbesondere wenn Sie einen kostenpflichtigen Dienst bzw. eine Anwendung anbieten möchten, müssen Sie bei einer Webanwendung eine für Sie und den zukünftigen Kunden recht umständliche Zahlungsabwicklung implementieren, während die Kreditkarte des Kunden bei iTunes und einer nativen Anwendung quasi einen Klick entfernt ist.

Dennoch: Mit immer mehr Browsern, unabhängig vom Betriebssystem, die die neuesten Standards, wie beispielsweise HTML5 und CSS 3 unterstützen, kann eine Webanwendung auf allen mobilen Plattformen laufen und Nutzer erreichen, egal ob diese ein iPhone, ein Android-Handy, ein Web-OS-Tablet oder ein Nokia-Telefon benutzen. Webseiten sind in der Regel viel einfacher und günstiger zu programmieren als native Programme. Viele Firmen beschäftigen bereits Heerscharen von Webentwicklern, die HTML, CSS und JavaScript beherrschen.

Wie jede andere Webseite im Internet muss eine Webanwendung auch keine Genehmigungsprozedur durchlaufen, bevor sie auf einem mobilen Gerät angewendet werden kann. Der Webseitencode wird in das Internet gestellt und ist sofort weltweit auf jedem Gerät mit einem geeigneten Browser verfügbar. Sie müssen keine Zertifizierungsgebühr bezahlen, keine zusätzliche Software kaufen, keine komplette Entwicklungsumgebung erlernen und können Aktualisierungen Ihrer Webanwendung in Sekundenschnelle weltweit veröffentlichen.

Warum nicht beides?

Dieses Buch zeigt Ihnen auf, wie Sie eine moderne mobile Webanwendung mit HTML5, CSS und JavaScript erstellen, die sich in vielen Punkten bereits im Browser wie eine richtige »native« Anwendung verhalten kann. Gleichzeitig wird Ihnen gezeigt, wie Sie aus dieser Web-App eine richtige »native« Anwendung für den iPhone App Store und den Google Play Store bauen können.

Das gibt Ihnen die Freiheit, Dienste zu erstellen, die für jedermann erreichbar sind, verbaut Ihnen aber nicht die Chance, mit der gleichen Anwendung auch in einen der App-Stores zu gelangen und dort bekannt zu werden. Idealerweise können Sie eine umfassende Webanwendung bauen, die Sie ständig (ohne umständliche Genehmigungen) aktualisieren, ausbauen und einen Teil als native App in verschiedene Stores stellen können.

Wer sollte das Buch lesen?

Das Buch richtet sich an alle, die sich für neuartige Webanwendungen für den mobilen Markt interessieren oder sich dort bereits bewegen und aus mobilen Webseiten leistungsfähige native Anwendungen erstellen möchten. Sie finden im Buch allerhand Codebeispiele, doch es richtet sich nicht nur an Entwickler, sondern auch an Produktmanager, Designer, Vordenker, Entrepreneurs und alle anderen, die sich mit dem neuesten Trend im mobilen Bereich auseinandersetzen möchten. Entwickler werden an dem Buch ebenfalls ihre Freude haben, da es einerseits ein Verständnis für den mobilen Markt außerhalb der Technik vermittelt und andererseits geballte HTML5-Beispiele auflistet, die helfen, mobile Seiten mit einer Funktionsfülle auszustatten, die bisher nur von »richtigen« Anwendungen bekannt waren.

Besonders tiefe Vorkenntnisse im Bereich der Programmierung setzt dieses Buch nicht voraus. Natürlich sollten Sie wissen, wie man einfache Webseiten baut und woraus sie bestehen. Begriffe wie HTML, CSS und JavaScript sollten Ihnen geläufig sein – Sie müssen sie aber nicht bis ins letzte Detail verstehen, um das Buch zu lesen.

Das Ziel des Buchs ist es, Ihnen die verschiedenen Möglichkeiten aufzuzeigen, die heutzutage mit HTML5 auf mobilen Geräten machbar sind und, falls Sie das auch noch interessiert, wie Sie sie nutzen können. Es vermittelt Ihnen einen Überblick über den mobilen Markt mit seinen Schwierigkeiten und Vorzügen, es zeigt Ihnen auf interessante Art und Weise, wie Sie möglichst viele mobile Plattformen mit wenig Aufwand abdecken und dazu noch in den wichtigen App-Stores wie dem iPhone App Store und dem Google Play Store vertreten sein können.

Wenn Sie noch tiefer in die Welt von HTML, CSS und JavaScript vordringen oder sich eingehender mit der nativen App-Entwicklung für mobile Plattformen oder mit mobilen Konzepten beschäftigen wollen, empfehlen wir Ihnen die Lektüre der folgenden Bücher, Blogs und Podcasts.

Heike Scholz (Hrsg.)
Björn Krämer / Torsten Schollmayer / Patrick Völcker

Android-Apps entwickeln
Konzeption, Programmierung und Vermarktung

Vorwort

Mobile Applikationen oder auch Apps haben sich für uns zu unentbehrlichen Begleitern entwickelt. Kaum eine Marke, kaum ein Unternehmen kann es sich heute leisten, nicht wenigstens in den beiden großen App-Stores von Google und Apple vertreten zu sein.

Zu einer erfolgreichen App gehört mehr als nur die reine Programmierung, insbesondere da sich heute immer mehr Entwickler auch als Unternehmer mit eigenen Apps betätigen. Es gilt also, schon weit vor der Programmierung den Grundstein für den Erfolg einer App zu legen. Und spätestens, wenn sie fertig ist, beginnt das Marketing.

Drei Mobile-Experten, alle erfahrene Praktiker, beschreiben in diesem Buch Vorgehensweisen und Werkzeuge, die für eine erfolgreiche Android-App notwendig sind. In drei Hauptkapiteln beleuchten sie den Entstehungsprozess einer App und geben praktische Tipps. Kapitel 1 beschäftigt sich mit den Bedürfnissen des mobilen Nutzers, der Informationsarchitektur, Prototyping und Mobile-App-Analytics. In Kapitel 2 werden dann die ersten Schritte der Programmierung einer Android-App gegangen, von der Zusammenstellung der Werkzeuge bis zur Fertigstellung einer ersten App. Kapitel 3 ist ganz dem Marketing der Android-App gewidmet, von den verschiedenen Werbemöglichkeiten bis zur App-Store-Optimierung, von Monetarisierungsmodellen bis zur Erfolgsmessung.

Ich als Herausgeberin bedanke mich ganz herzlich bei den Autoren für ihr Know-how und die Zeit, die sie in dieses Buch haben fließen lassen.

Den Lesern wünsche ich eine spannende Lektüre. Wenn Ihnen das Buch einige neue Anregungen oder praktische Tipps gibt, hat es sein Ziel erreicht. Geben Sie uns gern Ihr Feedback, wir würden uns freuen!

Hamburg

Heike Scholz, Herausgeberin

Inhaltsverzeichnis

2

Programmieren für Android

2.1 Android-Code

Es wird höchste Zeit, dass wir uns mit der eigentlichen Entwicklung nativer Apps für Android beschäftigen. Nach der meist von Euphorie und Inspiration geprägten Phase der Konzeption folgt nun das eigentliche Handwerk, welches aus der Idee ein fertiges Produkt macht. Denn die beste Idee nützt nichts, wenn sie nicht umgesetzt werden kann.

Glücklicherweise ist die Programmierung von Apps kein Hexenwerk, auch wenn man sich ungewohnterweise zum Beispiel damit beschäftigen muss, wie eine Anwendung reagieren soll, falls mitten in der Nutzungsphase das Telefon klingelt, die Netzverbindung abbricht oder der Akku abschmiert.

Um ein bisschen Gefühl für die mobile Umgebung zu entwickeln, werden wir im Verlauf der folgenden Seiten nach einer kurzen Einführung in Android-Java, die Entwicklungsumgebung und die Projektdateien drei der am häufigsten anzutreffenden Anwendungsgebiete von Apps besprechen und programmieren, welche die typischen Elemente des jeweiligen Genres schon einmal besitzen: eine rudimentäre App zur Geolokalisierung, ein kleines Spiel und die Anbindung an ein soziales Netzwerk.

Ganz nebenbei erhalten Sie noch ein kleine Einführung in die Google-Maps-API und das Facebook-SDK, damit Sie auch garantiert den Mehrwert von mobilen Endgeräten für ihre Apps nutzen können.

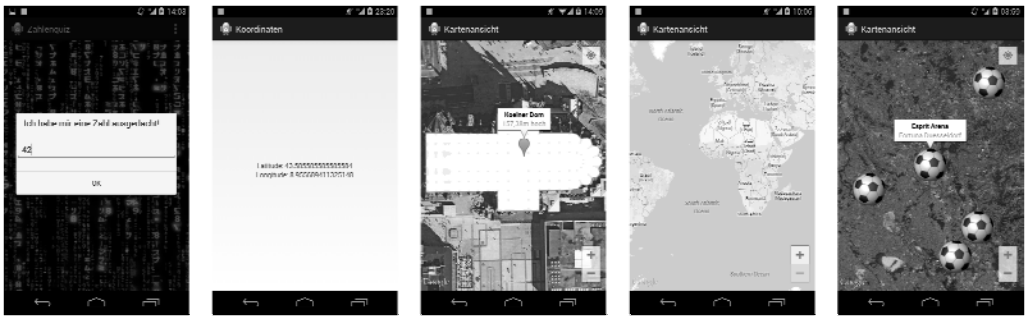


Bild 2.1: Die Apps im Überblick

Wenn Sie den Aufbau dieser Apps verstanden haben, können Sie sich problemlos daran machen, ihre eigenen Apps zu entwickeln. Allerdings empfehle ich bei einem ernsthaften Einstieg in dieses spannende Thema aufgrund der platzbegründet relativ oberflächlichen Behandlung der Programmierung in diesem Kapitel die Zuhilfenahme anderer Bücher, die sich eingehender mit Android-Code beschäftigen.

Auf den folgenden Seiten erhalten Sie in Form der Entwicklung erster kleiner Apps lediglich einen Grundstock mit auf den Weg, der Sie ein wenig in das Thema Programmierung einführen und Sie beflügeln soll, weiter an dem Thema zu bleiben, bis Sie Ihre eigenen Ideen entwickeln können.

Android-Architektur

Für dieses Buch ist es zu viel verlangt, auf die Details sowohl der Android-Plattform als auch des Betriebssystems an sich einzugehen. Ein paar Kleinigkeiten sollten Sie trotzdem wissen:

Wenn Sie eine App geschrieben haben, dann wird der Java-Code zuerst in Byte-Code kompiliert und anschließend mit einem Cross-Assembler für die Dalvik Virtual Machine (DVM) angepasst.

Wenn eine App nun auf einem Smartphone gestartet wird, dann springt – einfach gesagt – zuerst die Dalvik-VM an, lädt den zur App gehörigen Byte-Code und führt ihn aus, das heißt, der kompilierte Code wird nicht direkt auf dem Betriebssystem ausgeführt, sondern nur innerhalb der Grenzen der DVM.

Man fragt sich natürlich, warum nicht schon der zuerst für Java erzeugte Byte-Code verwendet wird. Dafür gibt es zwei plausible Gründe:

Der erste Grund ist rein lizenzrechtlich. Auch wenn die Sun Virtual Machine mittlerweile Open Source ist, so gelten für kommerzielle Produkte (was viele Apps ja sind) nach wie vor Restriktionen, die unter Umständen das Einholen kostenpflichtiger Lizenzen erfordern. Der Hauptgrund, warum Google den Compiler nachprogrammiert hat, ist, diese Einschränkungen zu umgehen, sozusagen als Dienst am Programmierer.

Der andere Grund ist die technisch unterschiedliche Herangehensweise: Während Sun mit Stacks arbeitet, ist Dalvik registerbasiert. Welche Arbeitsweise von beiden die leistungsfähigere ist, darüber scheiden sich die Geister. Entscheidend ist nur: Auch wenn die Programmiersprache im Prinzip sehr ähnlich ist, wird sie im Hintergrund – ohne dass wir es richtig merken – völlig anders vom Betriebssystem abgearbeitet.

Diese Werkzeuge zur Programmierung benötigen Sie unbedingt

Zu allererst einmal benötigen Sie eine ganze Menge Software, um sich die Entwicklung der Apps später zu vereinfachen. Neben dem Java Development Kit (JDK), um überhaupt Java programmieren zu können, brauchen Sie noch das Android Software Develpoment Kit (SDK), eine von Google zur Verfügung gestellte Sammlung von Tools für die Android-Entwicklung.

Welches Betriebssystem Sie für die Entwicklung benutzen – ob Windows, Linux oder Mac OS X – ist egal, die benötigten Tools stehen glücklicherweise für alle wichtigen Betriebssysteme zur Verfügung. Auf den folgenden Seiten werde ich Mac OS X verwenden. Der Grund hierfür ist, dass die meisten Entwickler für mobile Apps neben Android auch iOS bedienen wollen und dort für die Programmierung für das Apple-Betriebssystem leider nur eine Apple-Umgebung verwendet werden darf bzw. kann, gehe ich davon aus, dass das Entwicklungsbetriebssystem für die Android-Programmierung nicht gewechselt wird.

Aber keine Sorge, letztendlich liegen die Unterschiede hauptsächlich in der Installation der zu verwendenden Tools und in kleinen Details bei den Screenshots.

Neben den beiden obligatorischen Development Kits empfehle ich noch die Installation einer Entwicklungsumgebung (IDE). Zwar können Sie Ihren Code auch im Notepad, TextEdit oder anderen einfachen Texteditoren entwickeln und per Kommandozeile kompilieren, aber mit einer vernünftigen IDE, die auf die Belange eines Javaentwicklers spezialisiert ist, macht die Programmierung nicht nur doppelt so viel Spaß, die IDE hilft auch bei der Fehlersuche und beschleunigt Entwicklungszeit z. B. über Autocomplete der Codezeilen immens. Und für diesen Zweck ist die Open-Source-IDE Eclipse einfach State of the Art.

Damit Eclipse auch für Android-Entwickler alle Vorteile bietet, benötigen Sie als vorerst letztes Tool das Android Development Tools Plugin (ADT Plugin), welches die Entwicklungsumgebung mit dem Android SDK bekannt macht.

Sie können die einzelnen Werkzeuge nun entweder als komplettes Bundle unter

```
http://developer.android.com/sdk/index.html
```

herunterladen, was ich jedem Einsteiger empfehlen würde. Oder – falls Sie bereits Eclipse oder eines der anderen Tools im Einsatz haben – die Werkzeuge einzeln herunterladen und installieren.

Android Studio

Mutigen Lesern, denen es nichts ausmacht, dass sich ihre Entwicklungsumgebung stets weiter entwickelt und noch stellenweise abstürzen könnte, empfehle ich den Download des neuen Android Studio, welches Google dem überraschten Publikum bei der hauseigenen Entwicklerkonferenz I/O im April 2013 vorgestellt hat. Wo Sie dieses finden und wie sie es installieren, erfahren Sie am Ende dieses Kapitels.

Da sich das Tool zur Drucklegung des Buchs mit der Version 0.5.x noch im early access preview befand und sich die Beta-Testphase auch noch ein Weilchen hinziehen wird, werde ich innerhalb des ersten Projekts die einzelnen Schritte zum Programmieren von Apps jeweils für beide Entwicklungsumgebungen beschreiben.

Fairerweise muss man sagen, dass Android Studio in seiner Funktionalität auch noch eingeschränkt ist. Das Einbinden der Google Maps API zum Beispiel ist weitaus komplexer und bedarf viel manueller Arbeit, so dass ich für die Entwicklung unserer App Wo bin ich eher zu Eclipse raten möchte.

Installation des Java Development Kit

Da Android auf Java basiert, müssen Sie Ihren Rechner erst einmal für Java vorbereiten. Dazu müssen Sie das aktuelle Java Develpoment Kit (JDK) von Oracles Webseite herunterladen. Allerdings gibt es dort eine große Zahl an unterschiedlichen Versionen und auch wenn Java for Mobile Devices (Java ME) auf den ersten Blick verlockend ist, benötigen wir für die Android-Programmierung die Standard Edition (Java SE, und davon das JDK), die Sie unter

```
http://www.oracle.com/technetwork/java/javase/downloads
```

finden, herunterladen und installieren können.

Mit ein wenig Glück können Sie diesen Schritt allerdings auch überspringen, falls Sie die aktuelle Version bereits installiert haben. Das können Sie z. B. feststellen, wenn Sie auf die Webseite

```
http://javatester.org/version.html
```

gehen und die aktuelle Version angezeigt bekommen. Falls dort nichts angezeigt wird, können Sie auch

- unter Mac OS über Spotlight das Terminal öffnen und darin `java -version` aufrufen. Sie enthalten dann entweder die aktuell installierte Versionsnummer oder eine Meldung, dass Java nicht installiert sei.
- In Windows-Versionen vor Windows 7 können Sie die Version über denselben Aufruf auch in der Kommandozeile checken.
- Seit Windows 7 finden Sie in der Systemsteuerung das Java-Logo (die Java-Kaffeetasse). Sie erhalten dann die aktuell installierte Version angezeigt.

Installation des Android SDK

Etwas einfacher gestaltet sich die Installation des Android Software Development Kit. Sie finden es unter

```
http://developer.android.com/sdk
```

Nach dem Download müssen Sie die Datei entpacken und im neu erstellten Verzeichnis im Unterordner `/sdk/tools` die Datei `android` über *Öffnen mit > Terminal* den Android SDK Manager ausführen.

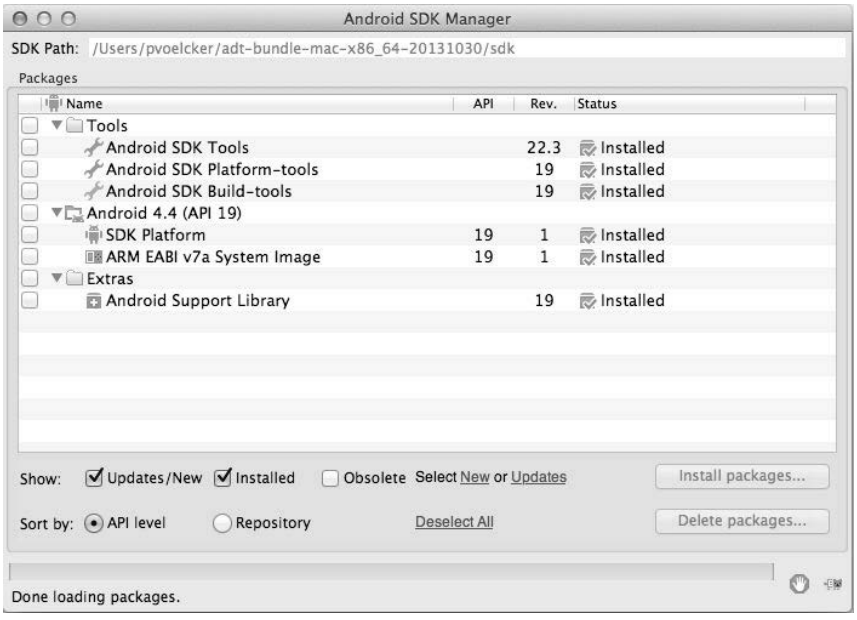


Bild 2.2: Android SDK Manager

Wenn Sie nur für die aktuelle Android-Version programmieren möchten, sollten die erforderlichen Packages bereits installiert sein.

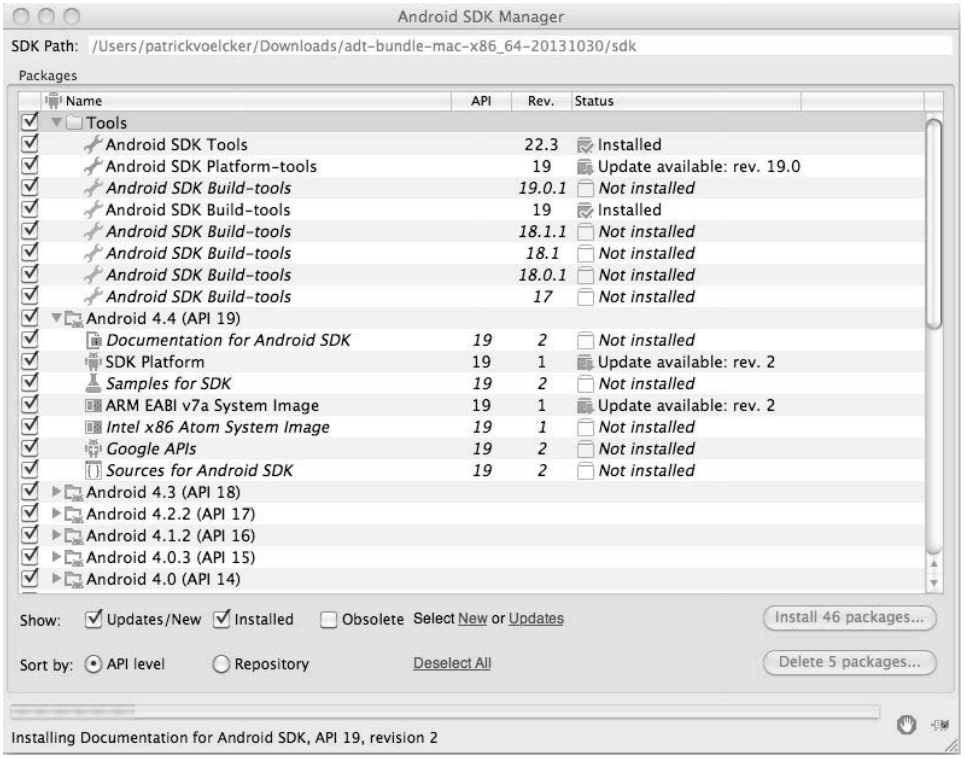


Bild 2.3: Der Android SDK Manager installiert die älteren Android-Versionen

Wenn Sie auch für ältere Android-Versionen entwickeln möchten, können Sie diese auch nachinstallieren. Der Android SDK Manager verbindet sich dann mit dem Google-Server und lädt nach einer Auswahl die benötigten Komponenten herunter. Da Gingerbread (Android 2.3) leider nach wie vor stark vertreten ist, empfehle ich diese und alle neueren Versionen (vielleicht mit Ausnahme von Honeycomb Android 3.1 und 3.2, wenn Sie nicht speziell nur für Tablets entwickeln wollen) zu installieren. Aber Vorsicht: Das kann eine ganze Weile dauern. Für den Anfang genügt es, wenn Sie nur die Android-Version herunterladen, die ihrem eigenen Testgerät entspricht.

Währenddessen können Sie die gewünschte Entwicklungsumgebung installieren, in diesem Fall Eclipse.

Installation von Eclipse – Die Entwicklungsumgebung

Letztendlich ist es egal, mit welcher Entwicklungsumgebung Sie Ihre Apps entwickeln wollen. Puristen nehmen Notepad oder TextEdit, Nerds greifen zurück auf Smultron, aber wenn Sie ein wenig Komfort wie Code Completion und Assistenten nutzen wollen, ist Eclipse nach wie vor die IDE der Wahl, und es ist auch die momentan von Google empfohlene Entwicklungsumgebung. Und natürlich ist auch sie kostenlos auf

```
http://www.eclipse.org/downloads
```

erhältlich. Laden Sie die *Eclipse Classic* oder die etwas kleinere, spezialisierte Version *Eclipse IDE for Java Developers* herunter, die Installation selbst sollte kein Problem darstellen.

Ursprünglich für (und auch mit) Java entwickelt, hat sich Eclipse zu einem mächtigen Hilfswerkzeug für zahlreiche Programmiersprachen entwickelt, die sich über Plugins einfach einbinden lassen können. Neben PHP, C, HTML und Python lässt sich somit auch Android-Code einbinden.

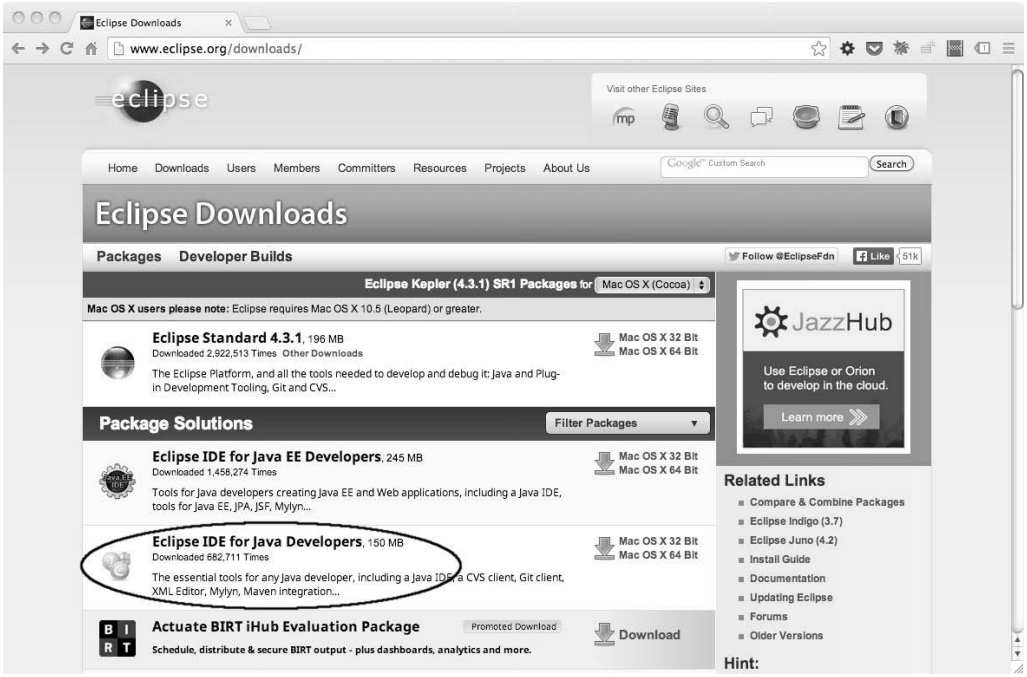


Bild 2.4: Die am besten geeignete Eclipse-Version finden

Warten Sie noch, bis der Android SDK Manager im Hintergrund seine Installationen vollendet hat, dann starten Sie Eclipse neu, definieren kurz das gewünschte Arbeitsverzeichnis (*Workspace*) und machen die Entwicklungsumgebung mit Android bekannt:

Bild 2.5: Bestimmen Sie hier das Arbeitsverzeichnis für die Eclipse-Daten.

Installation des ADT Plugins

Nur noch eine Installation, dann haben Sie es geschafft. Fügen Sie das Android Development Tools Plugin hinzu, in dem Sie in Eclipse unter *Help > Install New Software...* über *Add* die folgende URL

```
https://dl-ssl.google.com/android/eclipse/
```

ergänzen (und der Verbindung einen beliebigen Namen geben).

Dr. Dirk Koller

Android-Apps programmieren

Inhaltsverzeichnis

Danksagung

Meiner Frau Lisa möchte ich für das Anfertigen der tollen Skizzen danken.

Ein ganz besonderer Dank gebührt den Herren Marc Reichelt (*www.marcreichelt.de*) und Markus Gursch (*www.swordiApps.com*) für die fachliche Durchsicht.

Zu guter Letzt möchte ich Herrn Anton Schmid vom Franzis Verlag für die guten Tipps und Anregungen sowie die freundliche Zusammenarbeit danken.

Teil 3 – Erweiterung der Zeiterfassung

Mit Abschluss des letzten Kapitels ist die Zeiterfassung in ihren Grundzügen besprochen. Was nun in Teil 3 folgt, sind mehr oder weniger nette Erweiterungen, die dem Projekt etwas mehr »Glamour« verleihen. Klar, das muss natürlich auch sein. Aber diese Erweiterungen rütteln nicht mehr an der Architektur der Anwendung.

16 Location Based Services

Im Gegensatz zu einem Desktop-Rechner hat der Anwender ein mobiles Gerät wie ein Android-Smartphone immer in der Tasche. Weiß das Gerät, wo es sich befindet, dann wissen auch die Anwendungen, wo sich der Benutzer befindet. Zusammen mit einem Server, der Informationen über die nähere Umgebung hat, lassen sich damit wirklich faszinierende Anwendungen, sogenannte Location Based Services, erstellen.

Als praktisches Beispiel wird die Ortsbestimmung in der Zeiterfassung benutzt, um den Projektort zu speichern. Abhängig davon kann dann beim Erfassen neuer Zeiten das passende Projekt vorbelegt werden. Das funktioniert natürlich nur, wenn die Zeiten auch vor Ort erfasst werden.

16.1 Woher weiß das Gerät, wo es ist?

Die Antwort »GPS« auf diese Frage greift eindeutig zu kurz. Auch ohne freien Blick auf die GPS-Satelliten kennt das Gerät seine Position. Die Bestimmung erfolgt über drei verschiedene Wege.

16.1.1 Mobilfunksender

Die erste Möglichkeit zur Positionsbestimmung ist die Lokalisierung der Mobilfunkmasten in der Nähe. Das Gerät sucht in einer Datenbank nach den Masten, die es im Umfeld entdeckt hat. Aus den Standorten der Sender, die in der Datenbank hinterlegt sind, und der Entfernung zu diesen lässt sich die Position des Geräts mit einer Genauigkeit von ein bis zwei Kilometern bestimmen.

16.1.2 WiFi-Netzwerke

Die Positionsbestimmung mit Hilfe von WiFi-Netzwerken funktioniert ähnlich wie die Triangulation mit Mobilfunksendern. Allerdings werden hier die WLANs in der Umgebung zurate gezogen. Womöglich auch Ihres!

Man kann sich gut vorstellen, dass alle Mobilfunkantennen mit ihren Positionen in einer Datenbank zu finden sind. Wie aber kommt denn das private WLAN in eine Datenbank? Die Antwort klingt einigermaßen verwunderlich, stimmt aber: Mehrere

Unternehmen sammeln, mit oder ohne Unterstützung der WLAN-Besitzer, die Koordinaten aller WLAN-Router. Es macht dabei keinen Unterschied, ob das Gerät verschlüsselt ist oder nicht. Zumindest in Großstädten, wo die Dichte der Router sehr hoch und das Abfahren der Straßen schnell erledigt ist, kann mit Hilfe der so ermittelten Daten die Position inzwischen bis auf wenige Meter bestimmt werden.

16.1.3 GPS

Irgendwo auf dem freien Feld, abseits von allen WLANs und Mobilfunkmasten, hilft nur noch eine Technologie, das Global Positioning System, kurz GPS. Mit Hilfe des Empfängers für die Satellitensignale kann die Position bis auf wenige Meter genau bestimmt werden.

16.2 Location Services

Die Klassen zur Positionsbestimmung finden sich im Paket *android.location*:

- *LocationManager*
- *LocationProvider*
- *Location*
- *Criteria*

Die wichtigste Klasse des Pakets ist der *LocationManager*. Mit seiner Hilfe wird ein Provider ausgewählt und von diesem dann Location Updates angefordert. Eine Instanz der Klasse wird nicht etwa mit `new` erzeugt, sondern als *SystemService* angefordert:

```
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
```

Der *LocationManager* bekommt einen *LocationListener* zugeteilt. Es handelt sich dabei um eine Klasse, die das Interface *LocationListener* implementiert und auf Standort- und Statusänderungen reagiert. Wie bei *EventListenern* üblich kann man auf das explizite Anlegen der Klasse mit Namen verzichten, indem man eine anonyme innere Klasse verwendet. Die Klassendefinition wird dabei innerhalb des `new`-Operators eingeführt, ohne Angabe des Klassennamens:

```
LocationListener locationListener = new LocationListener() {

  @Override
  public void onLocationChanged(Location location) {
    Log.d(TAG, "Latitude: " + location.getLatitude());
```

```
        Log.d(TAG, "Longitude: " + location.getLongitude());
      }


      @Override
      public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
      }

      @Override
      public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
      }

      @Override
      public void onStatusChanged(String provider, int status, Bundle extras)
{
        // TODO Auto-generated method stub
      }
    } ;
```

Die Methoden innerhalb des *LocationListener*-Interface werden beim Eintreten der folgenden Ereignisse aufgerufen:

- *onLocationChanged*: wenn sich die Location ändert
- *onProviderDisabled*: wenn der Provider vom Benutzer deaktiviert wird
- *onProviderEnabled*: wenn der Provider vom Benutzer aktiviert wird
- *onStatusChanged*: wenn sich der Status eines Providers ändert

Damit haben wir alles beisammen um Location-Updates anzufordern. Ein erster einfacher (aber nicht wirklich empfehlenswerter) Aufruf sieht so aus:

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
```

Die Parameter zwei (*MinDistance*) und drei (*MinTime*) werden benutzt, um die Häufigkeit der Updates an den Bedarf der App anzupassen. Ist der Wert 0, werden so häufig wie möglich Updates angefordert (was in den seltensten Fällen sinnvoll ist).

Für eine Pollenflugvorhersage reicht eine Genauigkeit von Kilometern, für ein Navigationssystem dagegen nicht. Hintergrund hierfür ist, dass das Anfordern einer höheren Genauigkeit mehr Energie und Zeit kostet. Gleiches gilt für die Häufigkeit von Updates.

Die Angabe der Daten erfolgt in Millisekunden und Metern. Um also informiert zu werden, wenn sich der Benutzer 1 km entfernt hat oder 5 Minuten vergangen sind, sind zum Beispiel folgende Werte zu verwenden:

```
int minTime = 5 * 60000;
int minDistance = 1000;
```

Eine weitere Schwachstelle des ersten Aufrufs betrifft die fixe Vorgabe des `GPS_PROVIDER`. Es stehen mehrere Location-Provider zur Verfügung:

`GPS_PROVIDER`	Stellt GPS-Daten zur Verfügung
`NETWORK_PROVIDER`	Stellt WLAN und Funkzellen-Daten zur Verfügung
`PASSIVE_PROVIDER`	Nutzt Daten anderer Apps

Der GPS-Provider bietet zwar die höchste Genauigkeit, verbraucht aber auch am meisten Energie und benötigt sehr lange für die Ermittlung der Daten von mindestens vier Satelliten.

In vielen Anwendungsfällen ist es sinnvoller, Android den Provider auswählen zu lassen. Mit Hilfe der Klasse *Criteria* können die Anforderungen beschrieben werden.

Das folgende Codestück demonstriert die Vorgehensweise:

```
Criteria criteria = new Criteria();
criteria.setPowerRequirement(Criteria.POWER_LOW);
String provider = locationManager.getBestProvider(criteria, true);
locationManager.requestLocationUpdates(provider, minTime, minDistance,
locationListener);
```

Damit die Ortsbestimmung nicht unentwegt weiterläuft (und somit die Batterie des Geräts leert) sollte man schließlich unbedingt daran denken, den GPS-Provider mit Hilfe der Methode *removeUpdates()* wieder zu entfernen (in der Regel in der *onPause*-Methode).

Vor einem ersten Start fehlt noch eine letzte Kleinigkeit. Damit der Code, der ja immerhin Informationen aus der Privatsphäre des Benutzers ermittelt, auch ausgeführt werden darf, sind Einträge in der Manifest-Datei nötig (vor dem *application*-Tag):

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

`ACCESS_FINE_LOCATION` wird nur benötigt, wenn auf GPS-Daten zugegriffen werden soll.

Zur Vermeidung längerer Autofahrten beim Test des Codes zur Ortsbestimmung empfiehlt sich die Verwendung des Werkzeugs *Location Controls* des Dalvik Debug Monitors. Es findet sich auf der Registerkarte *Emulator Control* und erlaubt die Eingabe von Länge und Breite (siehe Bild 15.3) für den Emulator.

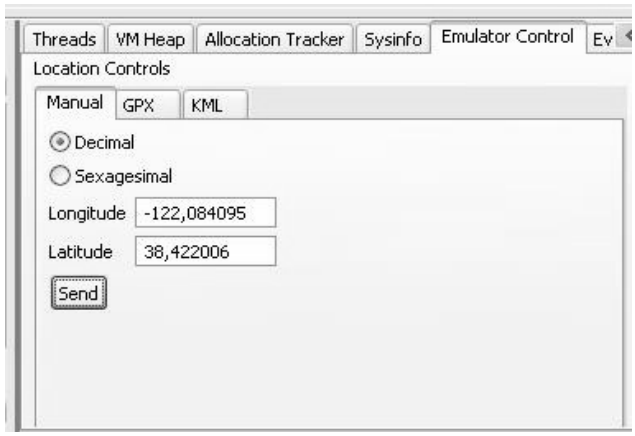


Bild 16.1: Simulation einer Position im Emulator

Mit den Reitern *GPX* und *KML* lassen sich komplette Bewegungsprofile simulieren. Entsprechende Daten erhält man zum Beispiel mit der Hilfe von Google Earth.

16.3 Koordinatenrechnereien

Bei der Erstellung von Location Based Services müssen oft Koordinaten in eine Adresse oder umgekehrt eine Adresse in Geokoordinaten umgewandelt werden.

16.3.1 Umwandlung einer Adresse in Koordinaten: (Forward-) Geocoding

Stellen Sie sich eine Anwendung vor, in der ein Außendienstmitarbeiter über ein Android-Tablet die Kunden herausfinden soll, die in der Nähe seiner aktuellen Position wohnen. Aus den Kundenadressen, die auf dem Server vorliegen, müssen nun Geokoordinaten gemacht und anschließend die Entfernung zwischen aktueller Position und der Kunden-Location in Form von Geokoordinaten berechnet werden.

Im Android SDK bietet die Klasse *Geocoder* ihre Dienste bei Realisierung dieser Anforderung an.

Die Klasse bekommt im Konstruktor neben dem obligatorischen Kontext ein Objekt vom Typ *Locale* übergeben, sodass die Ergebnisse in lokalisierter Form zurückgegeben werden können:

```
Geocoder geocoder = new Geocoder(this.getBaseContext(),
Locale.getDefault());
```

Die Methode zum Umwandeln einer Adresse in Koordinaten heißt *getFromLocationName()*. Sie liefert eine Liste von Adress-Objekten zurück und bekommt beim Aufruf neben dem Adress-String die maximale Anzahl an gewünschten Ergebnissen übergeben. Aus den erhaltenen Adress-Objekten können die Koordinaten ermittelt werden:

```
try {
   List<Address> addresses = geocoder.getFromLocationName("Unter den Linden
7, 10117 Berlin, Deutschland", 1);
   if (addresses.size() == 1) {
      Address address = addresses.get(0);
      Log.d(TAG, "Longitude:" + address.getLongitude());
      Log.d(TAG, "Latitude:" + address.getLatitude());
   }
}  catch (IOException e) {
   e.printStackTrace();
}
```

Ausgabe des obigen Codes in der LogCat-Konsole ist:

```
Longitude: 13.3962165
Latitude: 52.5166343
```

Für den oben geschilderten Anwendungsfall ist es natürlich sinnvoll, die Adressdaten bereits auf dem Server in umgewandelter Form vorzuhalten. Das Umwandeln größerer Adressbestände kann über spezialisierte Dienstleister erfolgen.

Wenn man den Geocoder verwendet, sollte man auch hier wieder bedenken, dass der Netzwerkrequest einige Zeit brauchen kann und der Aufruf deshalb in einem separaten Thread ausgeführt werden sollte.

16.3.2 Umwandlung von Koordinaten in eine Adresse: Reverse-Geocoding

Auch die Umwandlung in die andere Richtung wird oft benötigt. Denkbar wäre eine Anwendung, die Adressdaten eines Kunden automatisch speichert, wenn vor Ort die Position bestimmt wird. Aus den Geokoordinaten müssen also Straße und Ort ermittelt werden.

Auch hierbei hilft die Klasse *Geocoder*. Die entsprechende Methode heißt *getFromLocation()*. Sie liefert ebenfalls eine Liste von Adress-Objekten zurück und bekommt beim Aufruf neben *Longitude* und *Latitude* wieder die maximale Anzahl an gewünschten Ergebnissen übergeben. Das folgende Codestück demonstriert das Vorgehen:

```
Geocoder geocoder = new Geocoder(this.getBaseContext(),
Locale.getDefault());
try {
   List<Address> addresses = geocoder.getFromLocation(52.5172, 13.3949, 1);
   if (addresses.size() == 1) {
      Address address = addresses.get(0);
      Log.d(TAG, "Adresszeile 1: " + address.getAddressLine(0));
      Log.d(TAG, "Adresszeile 2:" + address.getAddressLine(1));
   }
}  catch (IOException e) {
   e.printStackTrace();
}
```

Ausgabe des obigen Codes in der LogCat-Konsole ist:

```
Unter den Linden 7
10117 Berlin
```

16.3.3 Entfernungsberechnung

Die dritte gängige Anforderung für Anwendungen mit Ortsbezug ist die Berechnung der Entfernung zwischen zwei Punkten. Typische Beispiele sind Apps à la »Restaurants in der Nähe«. Liegen alle Positionen erst einmal in Form von Geokoordinaten vor, lässt sich daraus mit ein wenig sphärischer Trigonometrie die Entfernung berechnen. Glücklicherweise ist die Formel zur Berechnung bereits im Android SDK enthalten. Die Klassenmethode *distanceBetween* der Klasse *Location* bekommt neben den Koordinaten der beiden Punkte noch ein Float-Array übergeben, in das die Ergebnisse geschrieben werden.

```
float startLatitude = 50.2250f;
float startLongitude = 8.8584f;
```

```
float endLatitude = 52.5166f;
float endLongitude = 13.3962f;
float[] results = new float[3];

Location.distanceBetween(startLatitude, startLongitude, endLatitude,
endLongitude, results);
Log.d(TAG, "Distance [m]: " + results[0]);
```

Manuel Di Cerbo / Andreas Rudolf

Android mit Arduino™ Due

Kommunikation zwischen Android-Geräten und Arduino™
Steuern Sie Ihren Arduino™ mit einem Android-Gerät
Praxisbeispiele zeigen, wie es geht

Inhaltsverzeichnis

4 Android und USB

Kernstück für die Experimente mit Android und Arduino ist die USB-Schnittstelle zwischen Android-Device und Arduino Due. Android verfügt über zwei Application Programming Interfaces (APIs), die es erlauben, über USB mit Peripherie zu kommunizieren.

Das Android-Device übernimmt dabei die Rolle des USB-Hosts und versorgt die Peripherie mit Strom. So ist es z. B. möglich, eine USB-Maus an ein Android-Gerät anzuschließen. Je nach Gerät ist hierzu allerdings ein USB-OTG-Konverter nötig. Android liefert dem Entwickler mit der USB-Host-API eine Möglichkeit, selber einen USB-Treiber für die angeschlossene Peripherie zu schreiben. Der Arduino verfügt über eine USB-Serial-Schnittstelle und kann daher via Android-USB-Host-API mit Android kommunizieren.

Die Android-Accessory-API bezweckt genau das Gegenteil. Die »Peripherie« agiert als USB-Host und versorgt das Android-Device mit Strom. Dabei wird der »native« USB-Port des Arduino Due mit einem USB-OTG-Konverter ausgestattet und das Android-Gerät angeschlossen.

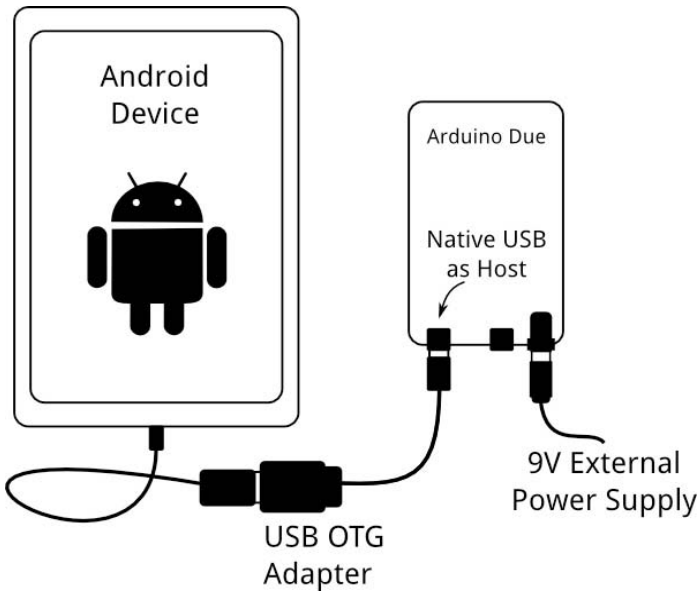


Bild 4.1a: Die USB-Kommunikation mit dem Arduino als Host.

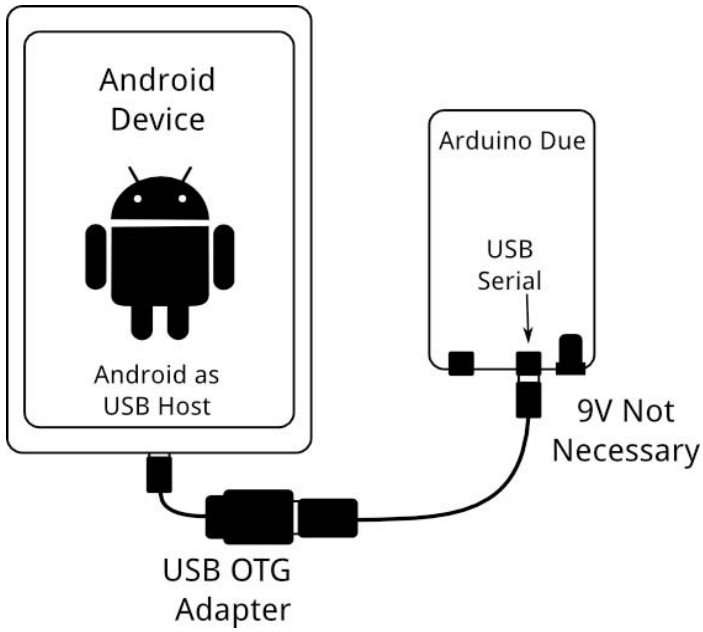


Bild 4.2b: Die Variante mit dem Android-Gerät als Host.

Lesezeichen
Auf der Entwicklerseite von Android findet man Dokumentation zu beiden APIs.
Android ADK:
http://developer.android.com/guide/topics/connectivity/usb/accessory.html
Android USB Host API:
http://developer.android.com/guide/topics/connectivity/usb/host.html

In den folgenden Abschnitten wird die Kommunikation zwischen Android und Arduino Due behandelt. Dazu wird das »Hello World«-Beispiel als Basis für beide Kommunikationsvarianten (Android-USB-Host-API und Android-Accessory-API) verwendet.

4.1 Kommunikation: Arduino als Accessory

Der Arduino soll als Android-»Accessory« betrieben werden. Dabei verwenden wir die Android-Accessory-API. Dazu teilen wir unserem Android-Manifest mit, dass wir eine Verbindung mit unserem Arduino erwarten. Sobald der Arduino verbunden ist, findet das Android-Betriebssystem Applikationen, welche auf das Anschließen eines Accessorys warten. Stimmen Modellname, Hersteller und Version des Accessorys mit dem im

Manifest erstellten »Filter« überein, kann so direkt eine Android-App gestartet werden. Zusätzlich wird der Activity vom System eine Schnittstelle zur Hardware via Argument übergeben. So kann die Kommunikation beginnen.

Die folgende Vorgehensweise kann auch in der API-Dokumentation auf *http://developer.android.com* gefunden werden.

Kommunikation über die Accessory-API
Es gibt zwei verschiedene Möglichkeiten, die Kommunikation via Accessory-API aufzunehmen. Unterschiedlich ist, wie die »Berechtigung« zur Kommunikation mit dem Accessory gehandhabt wird.

- Via Manifest-Eintrag:
Sobald der Arduino angeschlossen wird, erscheint eine Meldung vom System, die der Benutzer verwenden kann, um die Activity zu starten oder den Vorgang abzubrechen. Wird die Activity gestartet, so ist die Berechtigung bereits erteilt worden.

- Via Enumeration:
Während der Laufzeit einer App kann der Entwickler die vorhandenen ADK-Geräte vom System auflisten lassen. Bevor allerdings eine Kommunikation beginnen kann, muss die App den Benutzer nach der Berechtigung fragen.

Im Folgenden werden wir uns inhaltlich einfachheitshalber auf die erste Variante via Manifest-Eintrag beschränken.

Um einen Filter für unser Accessory zu erstellen, benötigen wir eine neue Datei *res/xml/accessory_filter.xml*. Der Ordner *xml* unter dem *res*-Verzeichnis ist standardmäßig nicht vorhanden und muss erst erstellt werden.

Den Inhalt des XML-Files schreiben wir folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<resource>
    <usb-accessory model="HelloWorldModel"
                   manufacturer="Hello Inc" />
</resource>
```

Diese Werte müssen mit den Definitionen im Arduino-Sketch übereinstimmen, damit das System erfolgreich die zugehörige Activity finden kann.

Als Nächstes referenzieren wir diese Meta-Daten in der Manifest-Datei von Android (*AndroidManifest.xml*). Hierzu wird ein neues *IntentFilter* für unsere HelloWorld-Activity erstellt.

```
<intent-filter>
    <action android:name=
     "android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
</intent-filter>
```

Das obige *IntentFilter* wird der Activity hinzugefügt.

Direkt anschließend referenzieren wir die Meta-Informationen:

```
<meta-data android:name=
"android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
android:resource="@xml/accessory_filter"/>
```

Was wir noch zusätzlich ändern, ist die mindestens benötigte SDK-Version (`minSdkVersion`). Diese setzen wir auf 12 (Android 3.1). Weiter teilen wir dem System mit, dass wir das »accessory hardware feature« verwenden möchten. Dies stellt sicher, dass die App nur auf Geräten installiert werden kann, welche dieses Feature auch unterstützen.

Die komplette *AndroidManifest.xml* sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.helloworld"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="12"
    android:targetSdkVersion="17" />

  <uses-feature
  android:name="android.hardware.usb.accessory"/>

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.helloworld.MainActivity"
```

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

      <intent-filter >
        <action
        android:name="android.hardware.usb.action.
          USB_ACCESSORY_ATTACHED" />
      </intent-filter>
      <meta-data
      android:name="android.hardware.usb.action.
        USB_ACCESSORY_ATTACHED"
      android:resource="@xml/accessory_filter"/>
    </activity>
  </application>
</manifest>
```

Wird nun ein Arduino angeschlossen, auf dem ein Android-Accessory-Sketch läuft, so erscheint ein Dialog, der den User darauf aufmerksam macht, dass eine App mit dem angeschlossenen Accessory kommunizieren möchte.

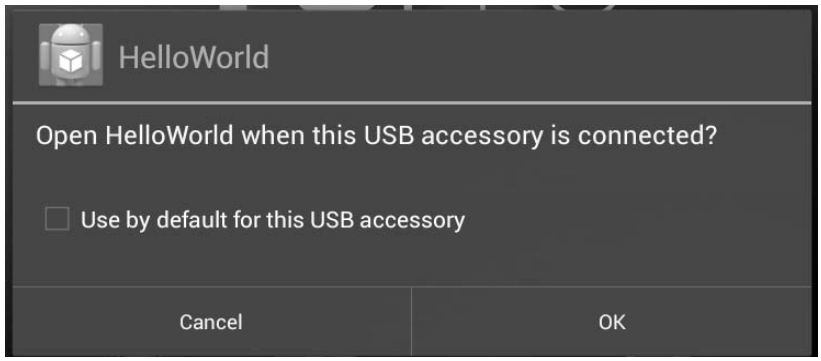


Bild 4.3: Die Dialogmeldung bei erfolgreich angeschlossenem Accessory.

Wird keine entsprechende Activity zum Accessory gefunden, so erscheint eine Dialogbox bei Anschließen des Arduino-Boards, die auf die in dem Arduino hinterlegte URL verweist.

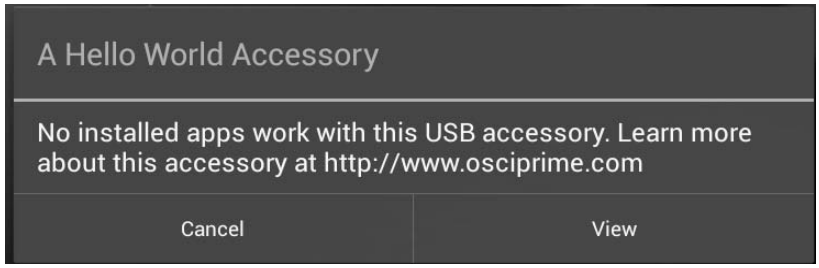


Bild 4.4: Diese Dialogbox erscheint, wenn keine Activity zum Accessory passt.

Beim Anschließen des Arduino Due wird ein »Extra« im Intent mitgeliefert, nämlich eine Referenz auf das Accessory. Mit dem Android-*UsbManager* erhalten wir so Zugriff auf das Accessory, um mit der Kommunikation beginnen zu können.

Die Android-Accessory-API nimmt uns sehr viel Arbeit ab. Tatsächlich ist das Accessory-USB-Protokoll eine komplexe Schnittstelle, welche Timing und Know-How von der Hardware benötigt. Die API nimmt uns die Initialisierung und die detailierte Kommunikation ab und ermöglicht das »Sprechen« mit dem Accessory über zwei simple Objekte. Dazu erhalten wir einen »FileDescriptor», in dem wir lesen und schreiben können.

```
FileDescriptor fd =
  mAccessoryFileDescriptor.getFileDescriptor();
FileInputStream fis = new FileInputStream(fd);
FileOutputStream fos = new FileOutputStream(fd);

...

fis.read(...); //receive something from the accessory
fos.write(...); //send something to the accessory
```

So erleichtert sich die Kommunikation mit dem Arduino erheblich und es wird ein rasches Implementieren der gewünschten Funktionalität ermöglicht.

Für unser nachfolgendes erstes Experiment »Hello World mit Android Accessory« senden wir dem Accessory ein »Byte« und regeln damit die Helligkeit der On-board-LED.

Beim Starten der Activity finden wir zuerst heraus, ob die Activity über den Launcher oder über das Anschließen des Accessorys gestartet wurde. Im letzteren Fall wird der Activity über den Intent, welcher die App startet, das Accessory als Argument mitgeliefert. In `onCreate` kann so direkt eine Referenz auf das Accessory-Objekt geholt werden.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
```

```
  setContentView(R.layout.activity_main);

  //see if we are started by ADK
  if(getIntent().hasExtra(UsbManager.EXTRA_ACCESSORY)){
    UsbAccessory accessory = (UsbAccessory)getIntent()
    .getParcelableExtra(
      UsbManager.EXTRA_ACCESSORY
    );
  }
  ...
}
```

Wird die Activity via Accessory-System-Dialog gestartet, so ist die Berechtigung auf das Device automatisch gewährt.

In der App wird nun eine Referenz auf das Accessory gespeichert, sodass beim Klick auf den Button eine Nachricht (Byte) gesendet wird. In unserem »HelloWorld« existiert bereits ein zweiter Thread, den wir etwas abändern.

Dem Klassenrumpf fügen wir eine Referenz auf das Accessory hinzu (`mAccessory`).

```
private UsbAccessory mAccessory = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Log.d(TAG, "onCreate in HelloWorld!");
  if(getIntent().hasExtra(UsbManager.EXTRA_ACCESSORY)){
    mAccessory = (UsbAccessory)getIntent()
      .getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
  }
  Button btStart = (Button) findViewById(R.id.btStart);
  btStart.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
      Log.d(TAG, "click!");
      if(sTaskThread == null){
        sTaskThread = new Thread(mTaskRunnable);
        sTaskThread.start();
      }
    }
  });
}
```

Das Task-Runnable wird nun so abgeändert, dass wir überprüfen, ob ein Accessory vorhanden ist und ob wir die Berechtigung zur Kommunikation vom Benutzer erhalten haben. Sind diese Konditionen erfüllt, öffnen wir den *FileDescriptor* des Accessorys und senden ein »Byte« an den Arduino Due.

```
private final Runnable mTaskRunnable = new Runnable() {
  @Override
  public void run() {
    Log.d(TAG, "Task started");
    UsbManager usbman = (UsbManager)
      getSystemService(USB_SERVICE);
    if(mAccessory != null &&
       usbman.hasPermission(mAccessory)){
      try {
        ParcelFileDescriptor fd =
          usbman.openAccessory(mAccessory);
        FileOutputStream fos = new FileOutputStream(
          fd.getFileDescriptor()
        );
        fos.write(new byte[]{0xFF});//send a byte to arduino
        fos.close();
        fd.close();
      } catch (Exception e) {
        Log.e(TAG,"Problem while communicating with
            Accessory");
      }
    }else{
      Log.d(TAG, "No accessory present or no Permission");
    }
    sTaskThread = null;
  }
};
```

Wird nun der »Start/Stop«-Button gedrückt, so startet der Thread, welcher ein Byte an den Arduino sendet (in diesem Fall hexadezimal 0xFF, dezimal 255). Das »Hello World«-Projekt gibt uns die Architektur für einen asynchronen Thread schon vor und erleichtert die Implementierung daher um einiges.

Anstatt jedes Mal die Kommunikation beim Drücken des Buttons neu zu initialisieren, sollten wir das möglichst nur einmal tun.

Bytes in Java
Ganze Zahlen werden in Java immer als »signed integer« interpretiert, d. h. sie sind immer vorzeichenbehaftet.
Erhält man nun ein »Byte« aus einem ByteBuffer oder aus einem Array und will dieses »Byte« als unsigned interpretieren, so kann dies mit einer Bitmaske geschehen.
`int interpretedValue = bufferByte & 0xFF;`
Durch die bitweise Addition von `0xFF` werden nur die hintersten 8 Bit des »bufferByte« für die Zuordnung verwendet.

Beispiel:
Sendet der Arduino eine unsigned 128 (0x80), so kann dies signed oder unsigned interpretiert werden. Im Zweierkomplement (vorzeichenbehaftet) wird die Bitfolge 0x80 (0b10000000) als »-128« interpretiert.

```
byte byteFromArduino = (byte)0x80; //let's say we got
                                   //a byte from the Arduino
Log.d(TAG, "Value is "+byteFromArduino); //will result in
                                    //"Value is -128"!!
Log.d(TAG, "Value is "+(byteFromArduino & 0xFF)); //results
                                    //in "Value is 128"
```

Generell gilt in Java: Sobald »rohe« Bytes interpretiert werden, ist Vorsicht geboten!

Im Beispiel wurde demonstriert, wie mit dem `FileOutputStream` etwas an den Arduino versandt werden kann. Das Lesen von Daten wird ähnlich implementiert. Allerdings wird dazu das Gegenstück, ein `FileInputStream`, verwendet.

Ein `FileInputStream` besitzt die Methode `read`:

```
int read(byte[] b)
```

Dabei versucht der `InputStream`, einen Inhalt in den Buffer »`b`« zu schreiben. Der `InputStream` strebt an, so viele Daten wie möglich (maximal `b.length`) zu lesen, gibt allerdings die Anzahl Bytes, welche tatsächlich gelesen wurden, als Return-Value zurück.

```
InputStream is = new InputStream(fileDescriptor);
byte[] buffer = new byte[1024];
int len = 0;

len = is.read(buffer);
for(int i = 0; i < len; i++){
  //process buffer[i];
}
```

Obiger Code zeigt den typischen Einsatz der `read`-Funktion eines `InputStream`.

4.1.1 Hello World mit Android Accessory

Ziel ist es, eine App zu erstellen (basierend auf den Erkenntnissen der »Hello World«-App), welche dem Arduino Werte zwischen 0 und 255 sendet. Via »SeekBar« wird damit die Helligkeit der On-board-LED gesteuert.

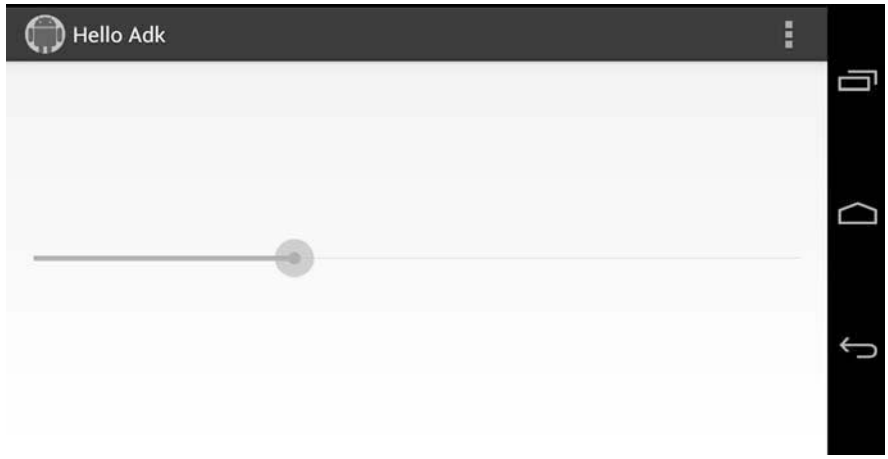


Bild 4.5: Der Bildschirm der »Hello Adk«-App.

Arduino-Sketch

Für dieses Beispiel wird auf dem Arduino Due der bereits bekannte Sketch verwendet, der die Helligkeit der On-board-LED je nach empfangenen Werten verändert. Der komplette Sketch ist hier noch einmal zur besseren Übersicht aufgelistet.

```
#include <adk.h>
#include <Scheduler.h>

#define LED 13
#define RCVSIZE 128

char model[] = "HelloWorldModel";
char description[] = "A Hello World Accessory";
char company[] = "Hello Inc";

char versionNumber[] = "1.2";
char serialNumber[] = "1";
char url[] = "http://www.osciprime.com";

USBHost Usb;
```

```
ADK adk(&Usb, company, model, description,versionNumber,url,serialNumber);

void setup()
{
  cpu_irq_enable();
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  delay(200);
  Scheduler.startLoop(adkLoop);
  Scheduler.startLoop(serialLoop);
}

void loop(){
   yield();
}

void serialLoop(){
   uint8_t cmd = 0;
   if(Serial.available()){
      cmd = Serial.read();
      analogWrite(LED,cmd);
   }
   yield();
}

void adkLoop()
{
  uint8_t buf[RCVSIZE];
  uint32_t nbread = 0;
  uint8_t cmd = 0;

  Usb.Task();

  if (adk.isReady()){
    adk.read(&nbread, RCVSIZE, buf);
    if (nbread > 0){
      cmd = buf[0];
      analogWrite(LED,cmd);
    }
  }

  yield();
}
```

Android-App

Die erste »Hello Adk«-App für die Accessory-API baut auf vier Files auf, die von unserer »Hello World«-App geändert werden:

MainActivity.java – Die Activity-Klasse für die App.

AndroidManifest.xml – Das Manifest, in dem Einstellungen für die App festgelegt werden.

res/layout/activity_main.xml – Layout für die Activity.

res/xml/accessory_filter.xml – Filter, das definiert, welche Parameter das Accessory aufweist.

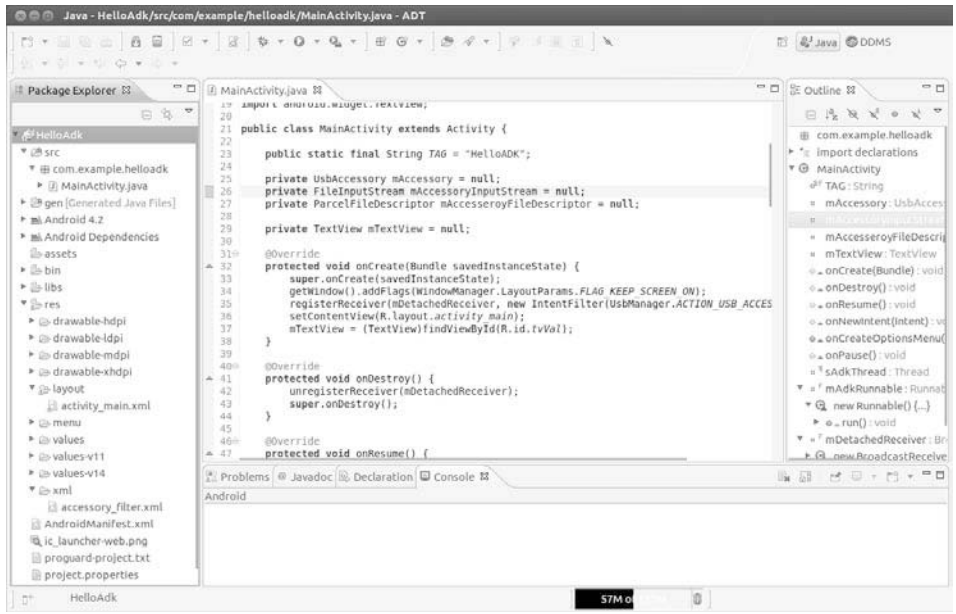


Bild 4.6: Die *MainActivity.java* unserer »Hello Adk«-App.

Im Folgenden werden diese vier Kernelemente der App betrachtet.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.helloadk"
  android:versionCode="1"
  android:versionName="1.0" >
```

Roland Willms/Yann Heeser

Spiele entwickeln für iOS und Android mit Cocos2D

Programmieren für die großen Smartphone-Plattformen
Entwickeln mit einem genialen Framework
Von der Spielidee bis zum App-Store

Vorwort

Cocos2D und Cocos2D-X

Cocos2D für iPhone ist ein Framework mit der Adresse

http://www.cocos2d-iphone.org/

Es dient zur Programmierung von 2-D-Spielen, ist aber auch für andere grafische und interaktive Applikationen nützlich. Es basiert auf dem Cocos2D-Design mit der Adresse

http://www.cocos2d.org/

und benutzt die gleichen Konzepte, ist aber in der Programmiersprache Objective C geschrieben. Neben den iOS-Geräten iPod, iPhone und iPad unterstützt Cocos2D für iPhone auch das Betriebssystem OS X.

Bild 0.1: Das Logo von Cocos2D

Cocos2D-X mit der Adresse

http://www.cocos2d-x.org/

ist eine Portierung von Cocos2D für iPhone in der Programmiersprache C++, um weitere Plattformen zu erschließen. Hierzu gehören mobile Systeme, zum Beispiel Android und Windows Phone, sowie Desktopsysteme, zum Beispiel Linux und Windows.

Bild 0.2: Das Logo von Cocos2D-X

Klassenspektrum von Cocos2D

Cocos2D ist einfach zu verwenden, integriert Praktiken von OpenGL ES unter iOS und Open GL unter OS X zur Optimierung der Schnelligkeit, hat eine aktive Benutzergruppe mit Forum, ist Open Source und kostenlos für eigene Spiele einsetzbar. Mehrere Tausend Spiele verwenden bereits das Cocos2D-API, darunter auch viele Bestseller.

Eine Liste verfügbarer Spiele finden Sie im Internet unter der Adresse:

http://www.cocos2d-iphone.org/games/

Cocos2D bietet zahlreiche Klassen für verschiedene Zwecke an: Szenenmanagement, Übergänge zwischen Szenen, Sprites, Effekte, Aktionen, Menüs, Schaltflächen, integrierte physikalische Engines, Teilchensysteme, Textunterstützung, Textur Atlas und Tile Maps, Soundmaschine, punktebasiertes Koordinatensystem zur Nutzung unterschiedlich großer Bildschirme einschließlich Retina-Bildschirme, Eingabe per Finger unter iOS und Tastatur unter OS X, Beschleunigungssensor, Hoch- und Querformat unter iOS, integrierte Pausen- und Fortsetzungsfunktion.

Am Beispiel des Projekts *Euro Crisis* beschreibt dieses Buch schrittweise die Entwicklung eines Spiels von einzelnen Elementen bis hin zum fertigen Programm. Hierbei kommen die wichtigsten Klassen von Cocos2D zum Einsatz.

Bild 0.3: Das Startmenü von Euro Crisis

Kapitel 1 befasst sich mit der Installation der nötigen Software für Cocos2D und Cocos2D-X, dem Start des ersten Programms und dem Überspielen der fertigen Software in die Stores. In Kapitel 2 erfahren Sie einige Grundlagen zum Spieldesign anhand konkreter Beispiele, um die Entwicklung eigener Ideen anzuregen. Von Kapitel 3 bis Kapitel 8 geht es um einzelne Elemente in Spielen: Bilder, Aktionen, Ereignisse, Sound, Beschriftungen und Teilchensysteme. In Kapitel 9 bekommen Sie einen Einblick in die innere Gestaltung von Spielleveln und die Koordination unterschiedlicher Stränge.

Bild 0.4: Koordination verschiedener Spielereignisse

Kapitel 10 und 11 behandeln den Rahmen um eine Spielhandlung. Hierzu gehören Szenen und Übergänge sowie Menüs und Schaltflächen zur Steuerung. In Kapitel 12 lernen Sie, wie einfach es ist, Spielereignisse und Bestenlisten lokal zu verwalten und auch global verfügbare Boards wie zum Beispiel das Game Center von Apple zu nutzen.

Objective C und C++

Obwohl Cocos2D-X nicht in der hauseigenen Sprache Objective C von Apple geschrieben ist, sind Vorbereitungen getroffen worden, dass die Programme auch unter iOS laufen. Daher müsste man eigentlich nur C++ beherrschen, um alle Plattformen mit Apps bedienen zu können.

Cocos2D für iPhone ist sehr früh auf den Markt gekommen, sodass Sie im Internet auf Websites und in Diskussionsforen meistens Quellcode in Objective C finden. Als das Framework immer häufiger in erfolgreichen Spielen eingesetzt wurde, begannen einige Entwickler damit, die Klassen auch in andere Sprachen zu portieren. Dies ist nicht einfach, weil jede Plattform spezifische Eigenschaften hat und die jeweilige grafische Umgebung berücksichtigt werden muss. So gab es für Android zunächst einen Port in Java und für Windows Phone einen Port in C#. Weil sich alle Plattformen jedoch technisch weiterentwickeln, blieb unklar, ob diese Ports in einigen Jahren noch laufen.

Mit der Übernahme der Entwickler von Cocos2D für iPhone in die Spielefirma Zynga ist sichergestellt, dass das Framework weiterhin gepflegt und an den neuesten Stand der Technik angepasst wird. Cocos2D-X ist mittlerweile relativ breit aufgestellt und wird sehr zeitnah nach Updates von Cococs2D für iPhone angepasst, sodass Sie davon ausgehen können, dass es in einigen Jahren noch laufen wird.

In diesem Buch finden Sie den Quellcode stets in zwei Varianten: zunächst in einer kommentierten Form in Objective C und immer direkt im Anschluss in unkommentierter Form in C++. So ist sichergestellt, dass Sie später beim Herumsuchen im Internet flexibel sind und Code in Objective C und C++ parallel verstehen.

Support

Das Spiel *Euro Crisis* als Programmierprojekt in diesem Buch und die Variante *Dollar Crisis* finden Sie in den Stores der jeweiligen Systeme.

Die Dateien für die Projekte in den einzelnen Kapiteln gibt es im Internet auf der Franzis-Seite:

http://www.buch.cd/

Hier finden Sie eine Anleitung, wie Sie mithilfe der ISBN dieses Buchs den Quellcode der Programme einschließlich der zugehörigen Ressourcen der Projekte erhalten.

Aus lizenz- und urheberrechtlichen Gründen sind die originalen Medien des Spiels wie zum Beispiel Icons, Bilder und Sounds in den angebotenen Dateien besonders gekennzeichnet oder ersetzt. Die mitgelieferten Medien dienen nur als Anschauungsmaterial im Rahmen dieses Buchs und dürfen nicht in eigenen Spielen verwendet werden. Im Internet stehen Ihnen zahlreiche Websites zur Verfügung, die Millionen Bilder und Sounds zur Lizenzierung für kommerzielle Zwecke anbieten und die Urheber entsprechend vergüten.

Für Diskussionen rund um das gesamte Klassenspektrum von Cocos2D gibt es das Forum mit der Adresse:

http://www.cocos2d-iphone.org/forum/

Es ist sehr gut besucht, sodass Benutzer rasch Antworten auf alltägliche Probleme bekommen. Mithilfe des Suchsystems können Sie leicht nachsehen, ob es bereits Antworten auf bestimmte Fragen oder Abhilfe bei aufgetauchten Fehlern rund um Cocos2D gibt.

Wir wünschen Ihnen viel Spaß beim Lesen des Buchs und der Programmierung Ihrer ersten Cocos2D-Apps sowie viel Erfolg bei der Vermarktung Ihrer Apps in den Stores.

Krefeld, im September 2012

Roland Willms — *http://www.cocos2d.de/*

Yann Heeser — *http://www.cocos2dx.de/*

Inhaltsverzeichnis

1 Hello Cocos2D

In diesem Kapitel behandeln wir:

- die Installation von Xcode und Cocos2D zur Programmierung einer App für iOS und OS X
- die Einrichtung einer App bei iTunes Connect mit den nötigen Informationen für den App Store
- das Anlegen eines Projekts für eine App in Xcode
- die Einstellung produktbezogener Angaben und die Integration eigener Icons und Startbilder in Xcode
- die Ausführung einer App in Simulatoren und auf angeschlossenen Geräten
- das Hochladen einer App in den App Store zur Verifizierung
- die Installation von Eclipse, dem Android SDK mit NDK und Cocos2D-X zur Programmierung einer App für Android
- das Erstellen eines Android-Projekts mit anschließendem Importieren in Eclipse
- das Hinzufügen eines eigenen Hintergrundbildes und die Integration eigener Icons
- die Ausführung einer App im Android-Simulator und auf Android-Geräten
- das Hochladen einer App in verschiedene Stores

1.1 Xcode und Cocos2D installieren

Zur Entwicklung von Apps unter iOS und OS X mithilfe von Cocos2D benötigen Sie die Entwicklungsumgebung Xcode von Apple und das Cocos2D-Framework.

1.1.1 Xcode installieren

Xcode ist eine Software von Apple zur Entwicklung von Programmen für die Betriebssysteme iOS (iPod, iPhone, iPad) und OS X (iMac, MacBook).

Starten Sie den App Store, suchen Sie nach »xcode« und installieren Sie die Software.

Bild 1.1: Xcode im App Store

Spätere Updates von Xcode können Sie jederzeit wieder über den App Store installieren.

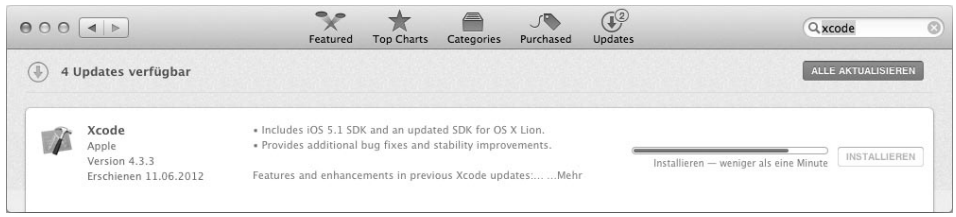


Bild 1.2: Updates im App Store

1.1.2 Cocos2D installieren

Besuchen Sie die Homepage von Cocos2D:

http://www.cocos2d-iphone.org/

Am 09.07.2012 ist nach mehr als einjähriger Entwicklung die Version 2.0 des Frameworks zur Entwicklung von 2-D-Spielen erschienen.

Im Download-Bereich finden Sie das komprimierte Archiv

cocos2d-iphone-2.0.tar.gz

mit der Cocos2D-Software. Es enthält einen Ordner mit den Dateien von Cocos2D. Kopieren Sie ihn auf den Desktop des Macs, und geben Sie ihm zur Vereinfachung den Namen:

cocos2d

Der Ordner enthält das Installationsskript:

install-templates.sh

Starten Sie *Finder/Programme/Dienstprogramme/Terminal* und geben Sie den Befehl

```
cd desktop/cocos2d
```

zum Wechseln in den Ordner cocos2d auf dem Desktop und den Befehl

```
./install-templates.sh -f
```

zum Installieren des Cocos2D-Frameworks ein.

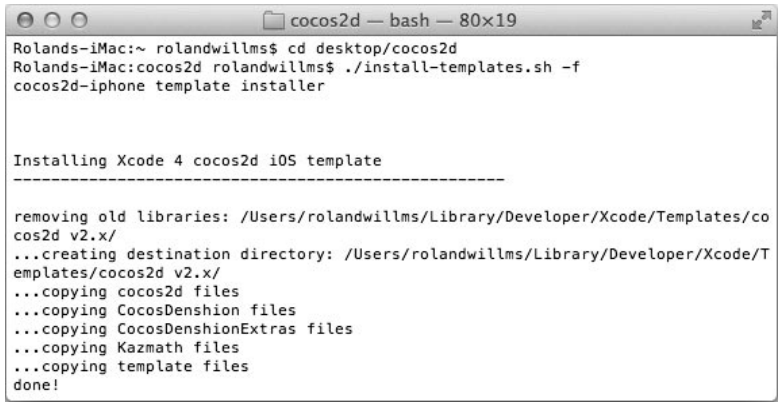


Bild 1.3: Installationsskript von Cocos2D

Das Skript kopiert die Dateien von Cocos2D in die Ordner

Library/Developer/Xcode/Templates/cocos2d v2.x

und

Library/Developer/Xcode/Templates/File Templates/cocos2d v2.x

1.2 Eine App für iOS erstellen

Auf der Homepage

https://developer.apple.com/

finden Sie Informationen von Apple zum iOS Developer Program (iPod, iPhone und iPad) und Mac Developer Program (iMac, MacBook).

Sie müssen Mitglied bei einem solchen Programm sein, um die entwickelten Apps auf realen Geräten testen und anschließend in den App Store hochladen zu können. Ohne eine solche Mitgliedschaft ist es nur möglich, Apps in den Simulatoren zu testen.

Im iOS Provisioning Portal finden Sie eine ausführliche Anleitung zur Installation eines iOS Development Certificate. Xcode benötigt ein solches Zertifikat, um den Test auf Geräten und das Hochladen in den App Store zu ermöglichen.

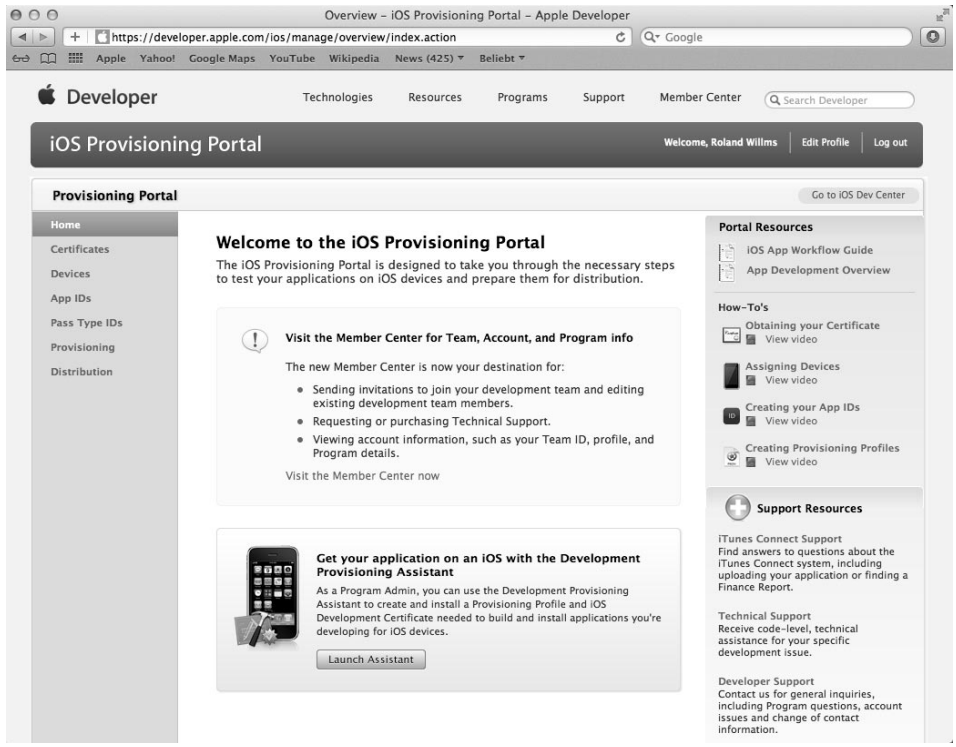


Bild 1.4: Informationen im iOS Provisioning Portal

Im Folgenden wird angenommen, dass Sie eine Mitgliedschaft besitzen und das Zertifikat zur Entwicklung von iOS Apps in Xcode installiert haben.

1.2.1 Eine App bei iTunes Connect einrichten

Loggen Sie sich bei iTunes Connect auf der Homepage

https://itunesconnect.apple.com/

ein. In Ihrem persönlichen Bereich finden Sie wichtige Links zu unterschiedlichen Themen.

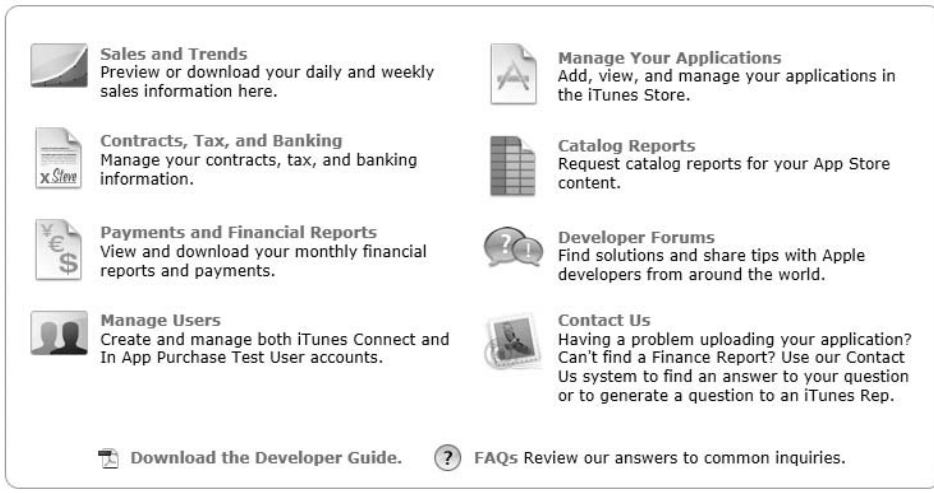


Bild 1.5: Wichtige Links in iTunes Connect

Sie sollten sich den angebotenen Developer Guide unbedingt herunterladen. Er enthält alle Informationen zu den Verträgen, zur Einrichtung, zum Hochladen und zur Verwaltung von Apps, zum Nachkaufen von Dingen mithilfe von In-App Purchases, zur Einrichtung des Game Centers, zur Verwaltung von Benutzerdaten in der iCloud und zur Werbung mithilfe des iAd-Netzwerks.

Um eine App einzurichten, folgen Sie dem Link *Manage your Applications*. Auf der nächsten Seite nutzen Sie die Schaltfläche *Add New App* und wählen auf der nächsten Seite den Typ *iOS App* aus.

Alle Angaben, die auf den folgenden Seiten gemacht werden, lassen sich später noch ändern, bis auf den App-Namen und die SKU-Nummer. Zunächst können Sie die Angaben grob eintragen. Bevor die App in den App Store hochgeladen wird, sollten aber alle Angaben vollständig ergänzt sein.

Auf der nächsten Seite geben Sie eine Sprache, einen App-Namen, eine SKU-Nummer und eine Bundle ID an. Der App-Name bleibt 180 Tage lang reserviert. Wenn nach dem Ablauf keine App hochgeladen ist, wird der Name für immer für den Account gesperrt und danach für andere Accounts freigegeben. Um den Namen zu schützen, können Sie einen Dummy als App hochladen und anschließend sofort wieder löschen.

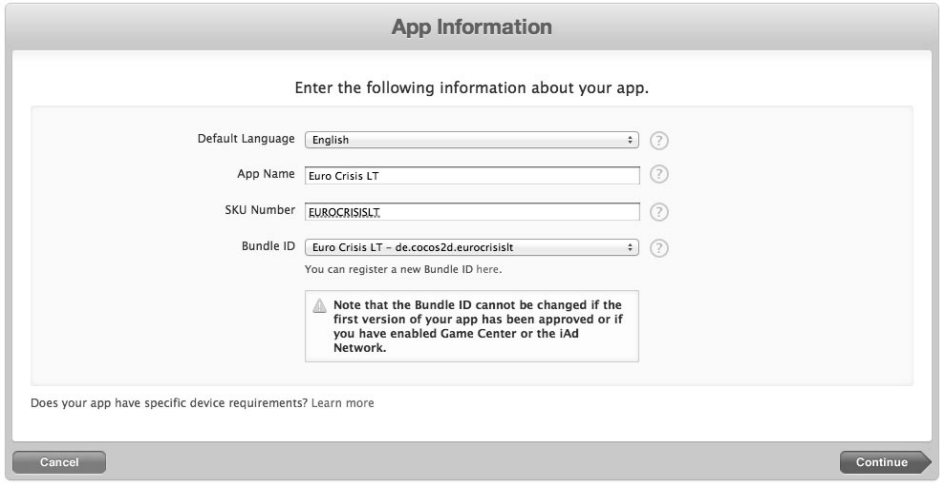


Bild 1.6: App Informationen

Die Bundle ID müssen Sie vorher in Ihrem Developer-Account angelegt haben. Wenn Sie auf den Link *here* klicken, gelangen Sie automatisch an die richtige Stelle und können dies nachholen.

Mithilfe der Bundle ID wird die App später weltweit identifiziert. Sie ist außerordentlich wichtig und sollte den allgemeinen Aufbau

```
<website>.<name>
```

haben, damit sie weltweit eindeutig ist.

Auf der nächsten Seite geben Sie das Verfügbarkeitsdatum an, ab dem die App im Store sichtbar sein soll, und stellen den gewünschten Preis ein.

Bild 1.7: Verfügbarkeitsdatum und Preis

Auf der nächsten Seite geben Sie noch einige Dinge an, zum Beispiel die Versionsnummer, die Beschreibung im Store und die Altersbeschränkung. Auch Icons und Screenshots werden hochgeladen, vorerst am besten weiße Bilder.

Bild 1.8: Icons und Screenshots

Die folgende Tabelle zeigt die Größen der einzelnen Bilder.

Bildtyp	*Größe*
Large App Icon	512 x 512 oder 1024 x 1024 (Retina)
iPhone / iPod Screenshots	960 x 640 oder 640 x 960 (Hoch- oder Querformat, beides Retina)
iPad Screenshots	1024 x 768, 768 x 1024 (Hoch- oder Querformat) oder 2048 x 1536, 1536 x 2048 (Hoch- oder Querformat, beides Retina)

Es empfiehlt sich, grundsätzlich nur noch Retina-Screenshots hochzuladen, um den perfekten Look auf den zugehörigen Displays zu bekommen. Andernfalls sehen die Informationen im Store auf den zunehmenden Retina-Geräten in einigen Jahren automatisch veraltet aus.

Nach der Bestätigung aller Angaben ist die App in iTunes Connect eingerichtet. Sie befindet sich nun im Status *Prepare for Upload.*

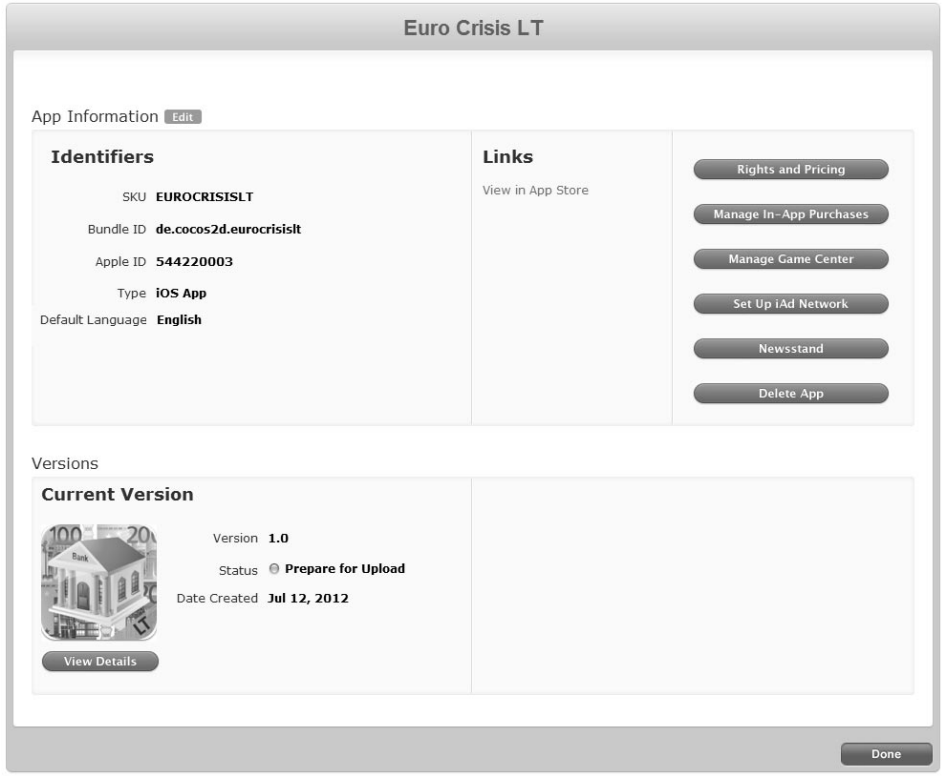


Bild 1.9: Informationen zur App in iTunes Connect

1.2.2 Eine App in Xcode anlegen

Starten Sie *Finder/Programme/Xcode* und rufen Sie das Menü *File/New/Project* auf. Unter iOS und Mac OS X befindet sich jeweils ein Eintrag *cocos2d v2.x* für das Cocos2D-Framework. Wenn Sie diesen Eintrag bei iOS auswählen, erscheinen drei Templates.

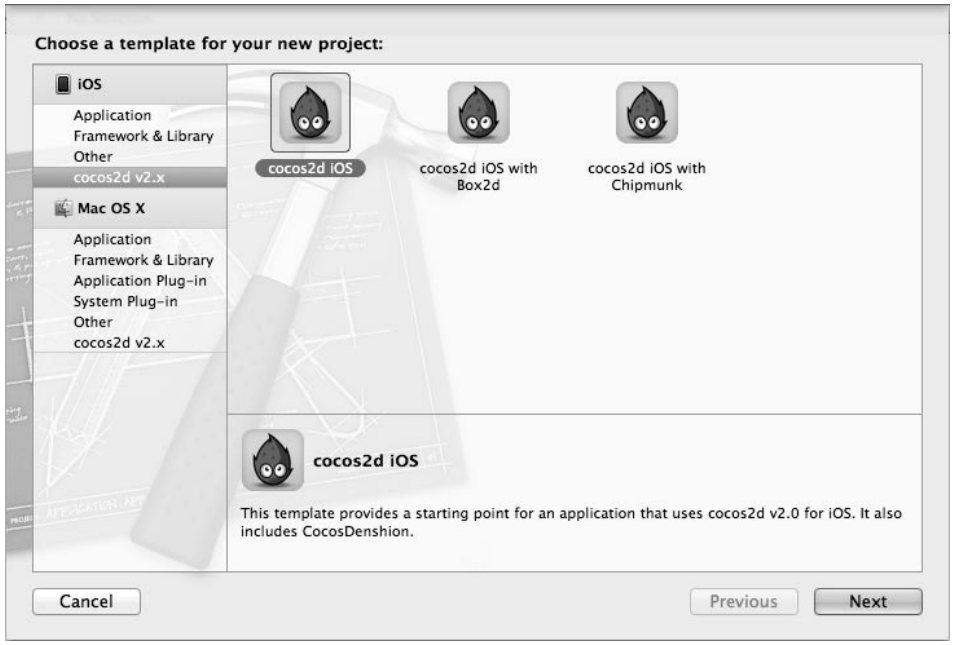


Bild 1.10: Auswahl von Cocos2D-Templates

Box2D und Chipmunk sind zwei sehr erfolgreiche Sets zur Entwicklung von Spielen, die mit physikalischen Eigenschaften wie zum Beispiel Schwerkraft und Mehrteilchensystemen arbeiten. Die Beschreibung dieser Sets ist im Rahmen dieses einführenden Buchs in Cocos2D nicht möglich.

Wenn Sie das erste Template *cocos2d iOS* auswählen, erscheint ein Dialog mit einigen Fragen.

Am wichtigsten ist die Angabe bei *Product Name*. Beachten Sie, dass dieser als Ordner- und Dateiname des Projekts verwendet wird. Insofern sollten Sie auf Leerzeichen und Umlaute verzichten. Im Screenshot ist *eurocrisislt* angegeben.

Bei *Company Identifier* sollte eine Website in umgekehrter Reihenfolge, also z. B. mit vorangestelltem *.de*. stehen, die auch tatsächlich registriert ist, um weltweite Überschneidungen zu vermeiden.