



Die Androiden-Toolbox

Kleiner Androiden-Führer
von
Andreas Itzchak Rehberg

[Version 39](#)

ANDREAS ITZCHAK REHBERG

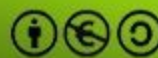


Die Androiden-Toolbox



AndroidPIT
Fill up your mobile

CC creative commons



INHALT

- 0. [Einleitung](#)
 - 1. [Vorwort](#)
 - 2. [Hinweise zur Nutzung des Buches](#)
- 1. [Sicherheit](#)
 - 1. [Safety](#) (Schutz vor Datenverlust)
 - 2. [Security](#) (Schutz vor Fremdzugriff)
 - 3. [App-Sicherheit](#)
- 2. [System](#)
 - 1. [Der Super-User "root"](#)
 - 2. [Dateisysteme und Datenstrukturen](#)
 - 3. [System-Info](#)
 - 4. [Konfiguration](#)
 - 5. [Automatisierung](#)
 - 6. [Dateimanager](#)
 - 7. [Androiden vom PC aus verwalten](#)
- 3. [Netzwerk](#)
 - 1. [Konfiguration](#)
 - 2. [Administration](#)
 - 3. [Sichere Übertragung](#)
 - 4. [WLAN](#)
 - 5. [Dienste bereitstellen](#)
- 4. [Tuning](#)
 - 1. [Einleitung und Überblick](#)
 - 2. [Generelle Tuning-Maßnahmen](#)
 - 3. [Längere Akkulaufzeiten erreichen](#)
 - 4. [Platz im internen Speicher schaffen](#)
 - 5. [Need For Speed](#)
- 5. [Anhang](#)
 - 1. [Fragen und Antworten](#)
 - 2. [Begriffserklärungen](#)
 - 3. [Google Permissions – und was sie bedeuten](#)
 - 4. [Akku-Verbrauch im Mobilfunk-Standby mit Tasker bekämpfen](#)
 - 5. [Diebstahlschutz mit Tasker](#)
 - 6. [Secret Codes oder Geheime Nummern](#)
 - 7. [Leistungsaufnahme verschiedener Komponenten](#)

Dieses eBook unterliegt einer [Creative Commons Lizenz](#). Du darfst es also auf jeden Fall in unveränderter Form weitergeben – und bei Quellen-Nennung zitieren. Weitere Details findest Du im verlinkten Lizenz-Text.



Die jeweils aktuellste Version dieses eBooks findest Du [hier](#).

Cover-Design: AndroidPIT

EINLEITUNG

Vorwort

Mein [AndroidPITiden-Buch](#) erfreut sich nach wie vor großer Beliebtheit, wie die Zugriffszahlen meines [Buchservers](#) ausweisen. Daher folgte ihm als zweiter Band [Mit Android auf Reisen](#), welches von Euch ebenfalls begeistert angenommen und mit Rufen nach "mehr" belohnt wurde. Was bleibt mir da anderes übrig, als für eine Fortsetzung zu sorgen?

Was diese zum Thema hat, habt Ihr ohnehin bereits am Namen des Buches erkannt: Tools. Also Werkzeuge. Die Teile, die sich im Werkzeugkoffer befinden – nicht die aus dem Strandkorb. Also nicht die Apps, mit denen man X oder Y nutzen kann – sondern die, mit denen man das System selbst im Griff hält, es konfiguriert und repariert. Ein Werkzeugkasten halt. Neben all diesen Apps sollen aber Praxis-Tipps und Hintergründe auch nicht fehlen.

Und woher mag wohl der Inhalt dieses Werkzeugkastens kommen? Überwiegend natürlich wieder aus meinen [App Reviews nach Einsatzzweck](#) bei [AndroidPIT](#), wie gewohnt. Und ebenfalls wie gewohnt, werde ich Euch laufend dorthin verweisen. Für mehr Details, oder für weitere Apps zu einem Themenbereich. Im Forum findet sich dann auch wieder ein Thread für Fragen und Anregungen. Oh Mann, so langsam schleift das ein...

Anders als im *AndroidPITiden-Buch* geht es in diesem Band aber um mehr als nur einen kurzen Überblick – hier steigen wir tiefer ein. Dennoch finden sich auch wieder Verweise zu den entsprechenden Themen im Forum. Nicht unbedingt zur Vertiefung, aber dort sind die Informationen aktueller und vollständiger. Alle Details bekommt man nie in ein Buch...

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](#) bist Du jederzeit herzlich willkommen! Auch für Dein Feedback sowie Deine Fragen zu diesem kleinen eBook findet sich dort ein Plätzchen.

An einigen Stellen wird auf eine "Franzis-Edition" dieses Buches hingewiesen, die "weitere Details" erhält – was den Einen oder die Andere Leser(in) verwundern könnte. Der Hintergrund ist jedoch leicht erklärt: Dieses zusätzliche Material entstammt einem Buch, an dem die Rechte für die deutsche Version beim Verlag liegen (*Android Forensics and mobile Security* von Andrew Hoog). Die genannten Inhalte wurden mir unter der Bedingung zur Verfügung gestellt, dass entsprechend detaillierte Ausführungen diesbezüglich exklusiv der Verlags-Version vorbehalten bleiben. Deal ist Deal 🤝

Zu guter Letzt noch ein kleiner technischer Hinweis: Sofern Dein eBook-Reader die StyleSheets korrekt unterstützt, erkennst Du verschiedene Arten von Links an ihrer Textfarbe: Rote gehen "[nach draußen](#)" (öffnen also wahrscheinlich Deinen Web-Browser), während grüne auf [Begriffserklärungen](#) im Anhang verweisen, und

auch blaue "[drinnen bleiben](#)" (also Querverweise innerhalb dieses eBooks sind). Des weiteren sind Ausführungen, die [root](#) voraussetzen, [entsprechend farblich hinterlegt](#).

Doch nun: Viel Spaß bei der Lektüre!

Hinweise zur Nutzung des Buches

Wie bereits im Vorwort aufgeführt, handelt es sich bei diesem Buch um die zweite Fortsetzung meines *AndroidPITiden-Buchs*. Ausgewählte Themen sollen hier vertieft werden. Für andere Dinge setze ich teilweise Kenntnisse voraus, die im "ersten Band" behandelt wurden.

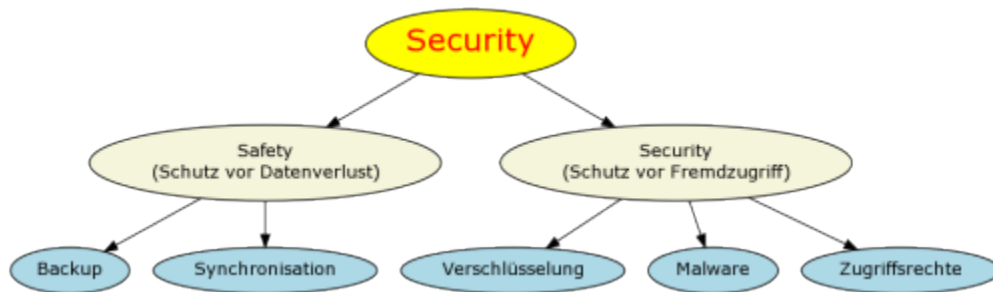
Auch wenn die Themen hier ausführlicher behandelt werden, so finden dennoch nicht alle verfügbaren Apps Erwähnung. Das ist auch gar nicht möglich. Nicht nur würde es den Umfang dieses Buches sprengen – es wäre auch schon nicht mehr aktuell, kaum dass ich es bereitgestellt hätte: Täglich finden hunderte 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 verlinkten 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...

SICHERHEIT

Leider haben wir im Deutschen dafür wohl nur das eine Wort – wo andere Sprachen, wie etwa das Englische, durchaus zu unterscheiden wissen. So hat jeder gewiss schon den Satz gehört: "Safety first!". An was denkt man dabei zuerst? Wahrscheinlich an Werbung für Präservative. Gutes Beispiel: Diese "Gummi-Tütchen" sollen u. a. vor unerwünschter Schwangerschaft oder der Ansteckung mit AIDS schützen. Doch verrät das gut sichtbare "Leucht-Kondom" im stockdunklen Schlafzimmer der Frau im Bett noch lange nicht, ob der nackte Typ da neben ihr wirklich der eigene Partner – oder vielleicht ein wildfremder Mann ist. Das wäre dann nämlich Security...

Also gilt es bei dem Wort "Sicherheit" sehr wohl zu unterscheiden, was man denn nun eigentlich meint. Und was ich hier meine, wenn ich diese Begriffe in die Tastatur hacke, soll folgende Grafik veranschaulichen – die damit gleich nebenbei die Struktur dieses Teils abbildet:



Safety

Schrieb ich da gerade "Safety first"? Nun, dann soll das auch für dieses Buch gelten. Mit "Safety" meine ich: Schutz vor Datenverlust. Schützt nicht vor dem Verlust des Androiden selbst: Genügend "Kleingeld" vorausgesetzt, lässt sich dieser noch relativ einfach ersetzen. Doch die persönlichen Daten, die gibt es nicht im Laden: Termine, Adressen, Fotos... Wo bekommt man die wieder her, wenn irgend so ein Depp die aus "Spaceballs" bekannte Ansage provoziert hat: "Vielen Dank, dass Sie den Selbstzerstörungsmechanismus aktiviert haben... 3...2..1..Tschüss!?"

Lokale Backups

Ah, aus dem Backup? Werfen wir zunächst einen Blick in "Murphys Gesetze". Da heißt es unter anderem:

Backup: *Etwas, das man nicht braucht, wenn man es hat – oder nicht hat, wenn man es braucht. Hat man es, wenn man es braucht, ist es entweder nicht lesbar – oder zumindest der Sektor mit der wichtigsten Datei kaputt.*

Damit einem dieses Schicksal erspart bleibt, kann man mit Netz, doppeltem Boden und Sicherheitsleine arbeiten: Ein Backup auf der SD-Karte, eine Kopie davon auf dem PC, und eine weitere außerhalb der eigenen vier Wände. Vielleicht nicht unbedingt im Bank-Tresor, aber auf einem fremden Server sollte es dann auch [verschlüsselt](#) sein. Nicht nur, um es vor unbefugten (Ein-) Blicken, sondern auch vor etwaiger Fremd-Modifikation zu schützen. Wie weit man das ganze jetzt treibt, bleibt natürlich jedem selbst überlassen.

Die nötigen Werkzeuge werde ich jedoch hier nennen. Ich muss dabei zugeben, dass ich in Sachen Datenschutz vielleicht ein wenig paranoid bin: So bevorzuge ich Lösungen, die ohne Netzwerk-Permissions auskommen. Sollen die Daten dann doch irgendwo außerhalb meines Androiden abgelegt werden, lasse ich das lieber von einer [separaten App](#) erledigen. Dieser Sachverhalt wird sich hier wohl ein wenig widerspiegeln...

Natürlich werde ich an dieser Stelle nicht auf alle verfügbaren Apps eingehen – dafür sind es einfach zu viele. Eine entsprechende Übersicht findet sich jedoch [im Forum](#).

Komplettsicherungen

Titanium Backup

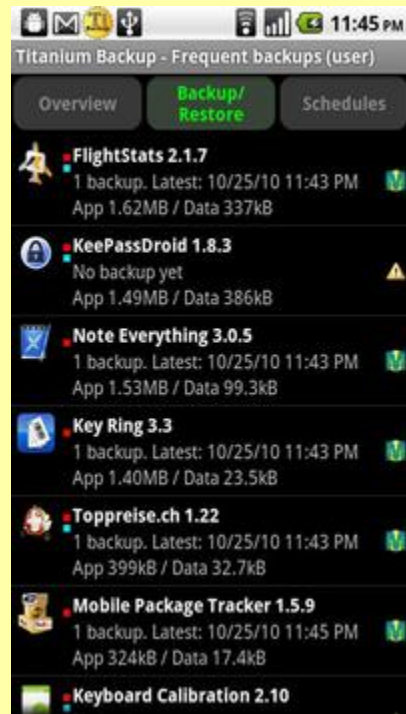
Fragt man nach einer guten Backup-App, taucht zweifellos [Titanium Backup](#) (Bild rechts) bei den Antworten auf. Aus meiner Sicht ist dies die wirklich beste Lösung, sie hat nur einen Haken: Man benötigt dafür root-Rechte. Bei genauerem Nachdenken ist das logisch – wie sonst will man wirklich *überall* rankommen? Und *Titanium Backup* kann wirklich alles sichern, System-Apps eingeschlossen. Letztere lassen sich damit sogar "einfrieren" oder, für ganz Harte, vollständig entfernen: So wird man die ganzen "Zwangsbeglückungen" los, die einem Hersteller (und, bei Geräten mit Branding, auch die Provider) auf's Auge drücken – die aber kaum jemand braucht.

Die Erläuterung aller Features dieser App würde beinahe für ein eigenes Buch ausreichen, also muss ich mich auf das Wesentlichste beschränken. Ein vollständiges Backup einschließlich aller Apps und ihrer Daten bedarf eigentlich keiner gesonderten Erwähnung – das kann man mit Fug und Recht erwarten (sonst wäre es ja kein Komplet-Backup; die Sicherung erfolgt übrigens wahlweise auf SD-Karte oder auch in die Dropbox). Gleiches gilt für die Wiederherstellung – denn was nützt ein Backup ohne diese Möglichkeit? Aber die zahlreichen Kleinigkeiten sind es, die *Titanium Backup* so einzigartig machen.

So lässt sich festlegen, wie viele "Generationen" man denn aufheben möchte – damit man bei Bedarf auch auf eine ältere Sicherung zurückgreifen kann. Eingangs erwähntem "Murphy" zufolge stellt man einen Fehler nämlich erst dann fest, wenn er schon ins Backup übernommen wurde. Oder der Market-Doktor: Taucht eine App im Playstore nicht mehr unter den eigenen Apps auf? Kein Thema: Der Market-Doktor repariert das. Hat ein Update eine App unbrauchbar gemacht? Na, dann holen wir uns aus einer "vorigen Generation" eine ältere Version zurück. Versehentlich in einer App zu viele Daten gelöscht? Die sind im Backup ebenfalls enthalten. Das Gerät wird immer langsamer? Dann lassen wir doch mal den [Dalvik](#)-Cache ein wenig aufräumen.

Und da wir ja alle ein wenig vergesslich sind – aber dennoch im Notfall gern ein aktuelles Backup hätten: Mit *Titanium Backup* lassen sich auch automatische Backups erstellen...

Für weitere Details etwa bezüglich Dropbox-Support, der Nutzung von Labeln, dem Wiederherstellen von Backups nach einem Systemupdate, oder gar dem Übertragen der Datensicherung auf ein neues Gerät, empfiehlt sich ein Besuch auf der [Homepage](#). Dort steht u. a. auch ein hilfreiches Wiki bereit.



Nandroid Backup

```
ClockworkMod Recovery v2.5.0.1
- reboot system now
- apply sdcard:update.zip
- wipe data/factory reset
- wipe cache partition
- install zip from sdcard
- nandroid
- partitions menu
- advanced

ClockworkMod Recovery v2.5.0.1
```

Eine weitere Art des Komplett-Backups führt über die sogenannte [Recovery-Konsole](#) – allerdings in der erweiterten Version, wie sie Custom-Recoveries bereitstellen. Am bekanntesten ist in diesem Umfeld ClockworkMod (kurz: CWM; links ein Screenshot einer älteren Version) von [Koushik Dutta](#) (besser bekannt als "Koush"). Für dieses findet sich im Netz sogar ein (zwar leicht veralteter, aber dennoch sehr nützlicher) [englisch-sprachiger Guide](#).

Anders als bei *Titanium Backup* werden hier keine einzelnen Dateien gesichert, sondern ein Image der gesamten Partitionen angelegt. Für Windows-User lässt sich dies vielleicht mit Norton Ghost vergleichen – eingefleischten Linux-/Unix-Nutzern reichen als Beschreibung die beiden Buchstaben dd. Aus jeder Partition wird somit eine Datei auf der SD-Karte, die sich dann natürlich auch zur weiteren

Sicherheit auf den Computer kopieren lässt.

Sinnvollerweise erstellt man in regelmäßigen Abständen – und insbesondere vor größeren Änderungen – eine solche Komplettsicherung, beispielsweise vor der Installation eines neuen ROM-Images. Geht dann etwas schief, lässt sich der "lauffähige Zustand" recht einfach wieder herstellen – indem man die erstellten Images über das Recovery-Menü wieder herstellt.

Und nun bringen wir die beiden genannten Kandidaten zusammen: Interessanterweise kann *Titanium Backup* nämlich auch einzelne Daten (etwa eine App inklusive ihrer Einstellungen) aus Nandroid-Backups wiederherstellen. Somit lassen sich ggf. kleinere "Fehler" wieder korrigieren – wenn man etwa vor dem Einspielen eines neuen [Custom-ROMs](#) die Daten einer App zu sichern vergessen hat. Oder sich erinnert, dass ein älteres Nandroid-Backup noch etwas enthält, was man auf dem neuen Gerät nicht mehr hat – wobei übrigens alternativ auch der [AppExtractor](#) helfen kann.

Wer so ein Nandroid-Backup für eine interessante Sache hält – es aber als zu umständlich empfindet, dafür jedes Mal ins Recovery-Menü zu booten (da ist man ja offline! Man könnte eine wichtige Facebook-Meldung verpassen!) – der sollte einmal bei den XDA-Developers vorbei schauen. Dort hat nämlich jemand ein Tool gebastelt, welches [Nandroid-Backups im laufenden Betrieb](#) erstellt. Also so ganz ohne Booten und ohne Facebook-Ausfall...

Komplett-Backups ohne root

Vom Androiden aus sind diese eigentlich unmöglich: Das Berechtigungs-System lässt es bekanntlich nicht zu, dass eine App auf die Daten einer anderen direkt

zugreift, ohne dass letztere diese explizit freigegeben hat – was in der Regel ja nicht der Fall ist. Wie aber will man ein solches auf anderem Wege erreichen?

Mit Android 4.0 (auch als *Ice Cream Sandwich* bekannt) wurde, nahezu undokumentiert, eine solche Möglichkeit geschaffen: Der auf dem Gerät laufende **ADB**-Daemon hat hier zusätzliche Rechte erhalten. Nutzen lassen sich diese für ein Backup allerdings nur, wenn auf einem per USB-Kabel angeschlossenen Computer auch ein entsprechender Client zur Verfügung steht. Dies trifft zu, sofern man das **SDK** dort installiert hat. Dann nämlich kann man den Befehl `adb backup` an der Kommandozeile absetzen:

```
adb backup [-f <file>] [-apk|-noapk] [-shared|-noshared] [-all] [-system|nosystem] [<packages...>]
```

Die erwähnten Optionen stehen dabei für folgendes:

- `-f` – Pfad und Name der auf dem Computer abzulegenden *.ab Datei. Bei dieser handelt es sich um eine komprimierte Archiv-Datei, welche die Apps und Daten des Android-Geräts enthält.
- `-apk` | `-noapk` – gibt an, ob die *.apk Dateien (Apps) mit verarbeitet werden sollen. Default ist `-noapk`.
- `-shared` | `-noshared` – Aktivieren/Deaktivieren des Backups von Inhalten des "externen Speichers" ("shared storage", SD-Karte). Default ist hier `-noshared`
- `-all` – gibt an, dass das gesamte System gesichert werden soll (Komplett-Backup). Alternativ lassen sich mit dem Filter `<packages>` auch einzelne (oder mehrere) Apps sichern.
- `-system` | `-nosystem` – spezifiziert, ob System-Apps (und ihre Daten) im Backup enthalten sein sollen. Default ist `-system`.
- `<packages>` – hier lassen sich einzelne Pakete (Apps) zur Sicherung angeben, wenn kein Komplett-Backup erfolgen soll. Bei Verwendung der Option `-all` wird dies nicht benötigt.



Ein Beispiel-Aufruf könnte wie folgt aussehen:

```
adb backup -apk -shared -all -f ~/backup/Desire_2012-08-28.ab
```

Oder, für Windows-Anwender:

```
adb backup -apk -shared -all -f C:\backup08262012.ab
```



Auf dem Gerät löst dieser Befehl sodann einen Hinweis aus, wie er im rechten Screenshot zu sehen ist. Der Cursor zeigt bereits an: Hier wird eine Benutzer-Eingabe erwartet. Und zwar soll das Passwort angegeben werden, mit dem das Backup sodann verschlüsselt wird; wir haben ja bereits gelernt: Das Wort "Sicherheit" hat zwei Bedeutungen. Durch das eigentliche Backup existiert eine Kopie, die vor Datenverlust schützen soll – und durch die Verschlüsselung wird Unbefugten der Zugang zu diesen Daten erschwert.

Analog geht es in der Gegenrichtung zu, wenn ein Backup wieder hergestellt werden soll:

```
adb restore ~/backup/Desire_2012-08-28.ab
```

Oder, für Windows-Anwender:

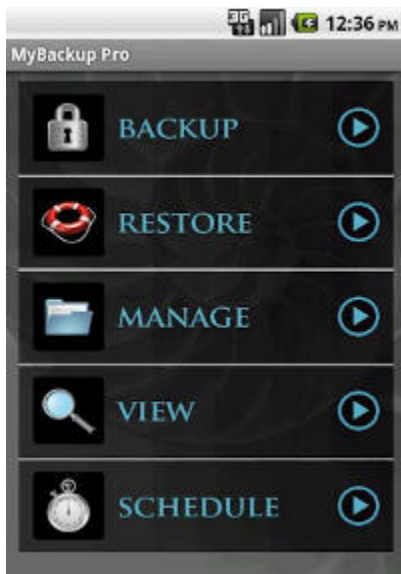
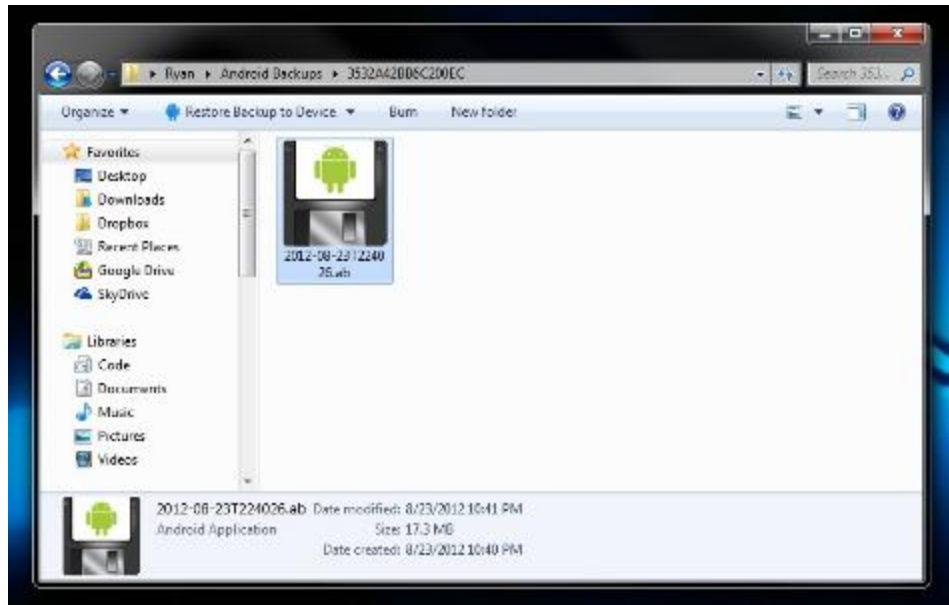
```
adb restore C:\backup\08262012.ab
```

Auch hier erfolgt wieder die Abfrage des Passwortes (linker Screenshot): Das Backup wurde ja verschlüsselt, und muss somit zur Wiederherstellung auch entschlüsselt werden. Wer beim

Wiederherstellungs-Befehl genauer hinschaut, hat es sicher bereits gesehen: Dieser kennt als einzigen Parameter die Datei des wiederherzustellenden Backups. Einzelne Daten lassen sich also scheinbar nicht aus einem vollständigen Backup direkt auf das Gerät zurücktransferieren.

Aber geht das wirklich nur über die Kommandozeile? Und von Hand? Die XDA-Developer haben sich da natürlich bereits etwas einfallen lassen. Und so berichtet ein dortiger Artikel vom [Ultimate Backup Tool](#). Natürlich für Windows. Und trotzdem wird es auch Linux-Fans und Mac-ianer freuen – handelt es sich dabei doch lediglich um eine einfache Batch-Datei. Da freue ich mich doch bereits darauf, dass diese bald als Shell-Skript auftaucht!

Was aber tun diejenigen, die viel lieber die Maus schubsen würden? Ein klein wenig warten, vielleicht. Denn Ryan (dem ich an dieser Stelle für den Hinweis auf diese Möglichkeit sowie auch für [seine Ausführungen](#) danken möchte) erweitert gerade seinen [Droid Explorer](#) (Screenshot unten). Ab Version 0.8.8.7 wird dieses Tool, welches sich sonst hauptsächlich an Wurzelmenschen richtet, dieses Backup auch für nicht-gerootete Geräte unterstützen. Weitere Tools, die dies unterstützen, sind übrigens auch bei den [Power-User-Tools zur Verwaltung des Androiden vom PC](#) beschrieben.



Eine echte Alternative zu finden, die sich direkt auf dem Androiden ausführen lässt, und dabei ebenfalls ohne *root* auskommt, scheint unmöglich. Nicht einmal wirklich nah heran kommt man da, wenn es etwas gutes sein soll. Die Bewertungen sind bestenfalls ziemlich durchwachsen, wie auch die Erfahrungen der Benutzer. Erschwerend kommen die Zugriffsrechte hinzu: Natürlich soll alles gesichert werden, auch Kontakte und Termine. Aber doch nicht das Internet...

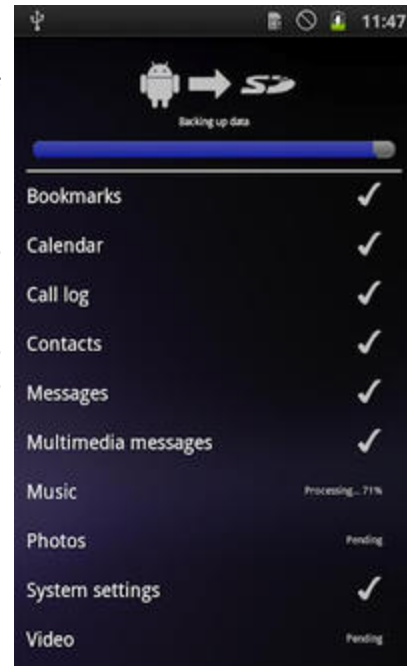
MyBackup Pro (linkes Bild) kommt in den Kommentaren noch recht gut weg (anders als die zugehörige Testversion). Die App soll ebenfalls ein komplettes Backup auf der SD-Karte anlegen, und natürlich von dort wiederherstellen können. Explizit erwähnt wird auch die Möglichkeit, auf diese Weise die Daten auf einem anderen Gerät wieder

einzuspielen.

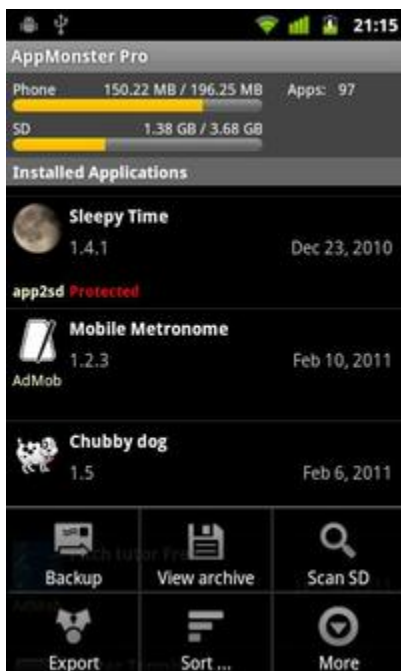
Da die App aber auch gern die Backups auf den Online-Servern des Anbieters speichern möchte, ist ein entsprechender Vertrauensvorschuss nötig: Die Kombination aus Zugriff auf Internet *und* persönliche Daten ist immer ein wenig heikel. Der Preis für diese App liegt mit knapp 4 Euro (eine gratis Testversion ist ebenfalls im Playstore verfügbar) leicht unter dem von *Titanium Backup*. Anwendern mit *root*-Zugriff würde ich dennoch eher letzteres empfehlen.

Auch bei [Sprite Backup](#) ist die Liste der verlangten Berechtigungen lang (für eine Backup-App ja normal), und schließt neben dem Zugriff auf persönliche Daten wie Kontakte auch das Internet ein: Eine Datensicherung ist hier auf die SD-Karte, aber auch in die Dropbox oder bei Box.Net möglich. Also in die Cloud – und das geht nun einmal nicht ohne Internet. Laut Screenshots scheint hier zwar fast alles gesichert zu werden – doch die Kommentare sind, euphemistisch ausgedrückt, recht durchwachsen, und lassen Zweifel an einer vollständigen Wiederherstellung aufkommen. Leider bietet die knapp vier Euro teure App auch keine Testversion, um sich selbst vor dem Kauf ein Bild machen zu können – in den 15 Minuten, die Google hier zum Testen gewährt, ist ein solcher schlicht unmöglich.

Und so erschöpfen sich die Alternativen auch schon – denn bei den anderen mir bekannten nicht-root Lösungen sieht es nicht besser aus. Bleibt als Fazit der Satz stehen: "Ohne root sich nichts tut". Zumindest im Bereich der Komplettbackups.



Apps sichern



Dafür sieht es in diesem Bereich auch für Anwender ohne root recht gut aus – denn hier gibt es das [AppMonster](#). Dieses schlägt in der Pro-Version mit knapp drei Euro zu Buche – hat mir aber nicht nur einmal den Tag gerettet.

Die Hauptfunktionalität dieser App besteht darin, bei jeder Neuinstallation automatisch eine Kopie der [APK-Datei](#) auf der Karte abzulegen – was sich natürlich auch manuell erledigen lässt. Erweist sich ein Update im Nachhinein als nachteilig, kann man so mit Leichtigkeit auf eine vorige Version zurückgreifen. Gleiches gilt nach einem Factory-Reset: Auch wenn die Lieblings-App aus dem Playstore verschwunden sein sollte, das Monsterchen hat eine Kopie aufgehoben.

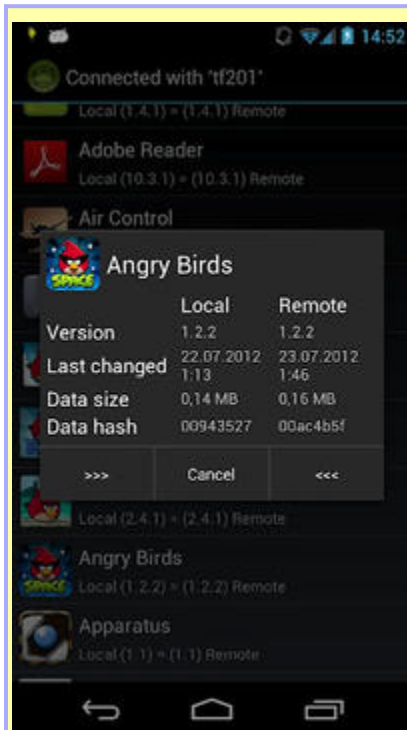
Da kann man es fast als "nette Dreingabe" betrachten, dass so einige Zusatzinformationen in der Appliste angezeigt werden: Etwa ob eine App mit Werbemodul (AdMob) daherkommt – was dann auch die Berechtigung für den Internet-Zugriff erklären könnte. Oder ob App2SD unterstützt wird. Des weiteren hat man auch direkt Zugriff auf die Playstore-Seite der jeweiligen App – oder kann sie über Facebook, Twitter, oder auch per Mail weiterempfehlen. Oder ganz anders herum: Sie schnell deinstallieren...

Fairerweise sei hier noch auf die Einschränkungen der kostenlosen Version hingewiesen: Keine automatische Sicherung der Neuinstallationen, kein Batch-Restore, und auch die erweiterten Informationen (wie App2SD) werden hier nicht angezeigt. Backups lassen sich aber manuell erstellen und auch wiederherstellen. Aus eigener Erfahrung kann ich den Kauf der Vollversion jedoch wärmstens empfehlen!

Was *Titanium Backup* im Komplett-Bereich, ist *AppMonster* hier: Mir ist keine App bekannt, die dem Monsterchen das Wasser reichen könnte. Alles andere kommt allenfalls "recht nah heran" – wie etwa das rechts abgebildete *OnTheFly Backup*, welches jedoch seit 2010 nicht mehr aktualisiert wurde. Es ist annähernd mit der Gratis-Version vom *AppMonster* vergleichbar: Alle Apps lassen sich manuell sichern, auch in mehreren Versionen. Zusätzlich unterstützt *OnTheFly* allerdings das Batch-Restore – ein Feature, dass bei *AppMonster* der Pro-Version vorbehalten ist.



Keine der anderen mir bekannten Apps in diesem Sektor unterstützt – wie die beiden genannten Kandidaten – das simultane Speichern mehrerer Versionen (sodass man bei Bedarf zu einer älteren Version zurückkehren kann). Dies halte ich jedoch hier für essentiell: Für alles andere habe ich ja, Dank *Titanium*, mein Vollbackup. Anwender ohne *root* sehen das vielleicht ein wenig anders: Für diesen Fall gibt es ja den bereits genannten *Forenlink*...



Ein spezieller Backup-und-Restore-Wunsch tritt meist dann ein, wenn das Amazon-Päckchen mit dem lang ersehnten neuen Androiden eingetroffen ist. Dann wird ausgepackt, herum gespielt, gerootet, die Konfiguration durchgeschaut, das Eine oder Andere ausprobiert... Alles selbstverständlich, während der "Alte" noch in Benutzung ist.

Und dann ist es soweit: "Da war doch noch was..." Genau: Wie kommen jetzt die Apps von A nach B? Klar, mit *Titanium Backup*. Also auf dem "Alten" alles sichern, von der Karte auf den Rechner kopieren, vom Rechner auf die Karte des "Neuen"... Besonders umständlich wird es dann, wenn beide Geräte noch länger parallel laufen müssen.

Aber warum schreibe ich das alles so dramatisch, wenn ich nicht mal wieder etwas anbieten möchte? Es muss also offensichtlich eine einfachere Lösung geben. Die trägt den Namen *AppSync*, und ist im Bild links zu sehen. Wo man auch erkennt: Die gewünschte App kann ruhig bereits auf beiden

Geräten vorhanden sein. *AppSync* muss das sogar. Und auf beiden Geräten mit dem selben Passwort konfiguriert, kann die Party beginnen: Kopiere von Lokal nach Remote. Oder umgekehrt. Neu nach Alt. Oder anders herum. Oder gar nicht (*Cancel*).

Und warum soll man sich dabei auf den "Umzug" beschränken? Da drängen sich förmlich weitere Anwendungsgebiete auf: Unterwegs in der S-Bahn mit einer App auf dem Smartphone gewerkelt – und bei Ankunft zu Hause den aktuellen Stand auf's Tablet übertragen, wo es dann bequemer weitergehen kann. Oder morgens noch schnell den Spielstand von Tablet auf das Smartphone schieben – damit man damit in der U-Bahn weiter daddelt, ohne dass der Spielstand leidet.

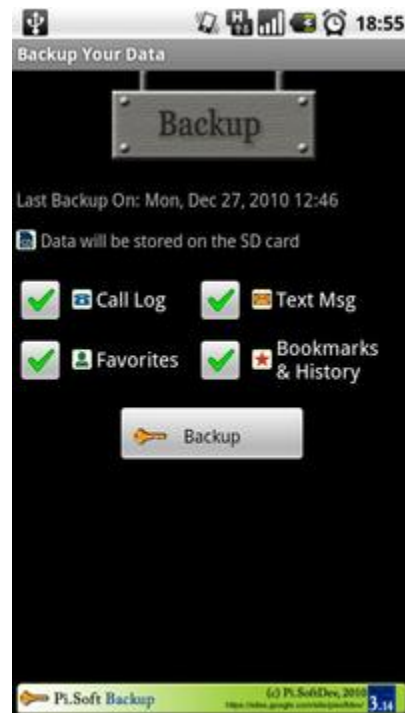
Ich bin mir sicher: Findige Geister wissen noch mehr Möglichkeiten. Und wen die Werbung stört, oder wer sich vor Begeisterung nicht mehr halten kann: Für ca. 4 Euro gibt es im Playstore den passenden "Pro-Key" zu erwerben...

Kontakte, SMS & Co.

Für die Kontakte ist die Frage nach einem lokalen Backup schnell beantwortet: Das ist bereits von Haus aus dabei, auch wenn es gern übersehen wird. Einfach einmal die Kontakte-App starten, die Menü-Taste drücken, und Export auswählen – und schon landet die gesamte Kontaktliste mit allen Details, einschließlich Fotos, als VCard-Archiv auf der SD-Karte. Dieses Format versteht natürlich auch die Import-Funktion derselben App – ebenso aber nahezu jede Anwendung auf dem PC, die sich um Adressen kümmert.

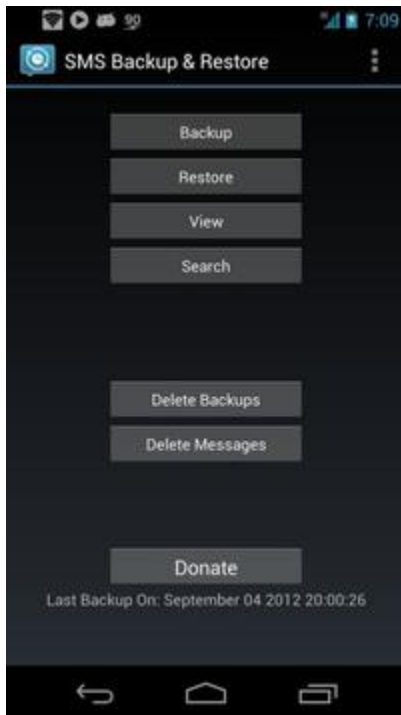
Ebenfalls bereits im Auslieferungszustand an Bord: Die Synchronisation mit Google. Wer kein Problem damit hat, seine persönlichen Daten im Netz abzulegen, kann auf diese Weise Kontakte und Termine sichern. Google scheint das als selbstverständlich anzusehen – denn wenn man die zugehörigen Einträge in der Konfiguration nicht selbst findet und abschaltet, ist dies automatisch aktiv. Aber der Synchronisation ist ein eigenes Kapitel gewidmet, daher soll hier die Erwähnung genügen.

Bei den eigenständigen Backup-Apps fange ich auch hier mit denen an, die möglichst vieles abdecken. Und da käme [Pi.Soft Backup](#) (Bild rechts) an erster Stelle: Die Vollversion kostet gut zwei Euro, und bereits die ebenfalls im Playstore verfügbare gratis Testversion (nur in der Laufzeit auf 15 Tage beschränkt, sonst voller Funktionsumfang) verzichtet auf die allgegenwärtige Internet-Berechtigung: Hier ist man sich offensichtlich der Sensibilität persönlicher Daten bewusst – was für mich und meine Mit-Paranoiker eine erfreuliche Tatsache ist.



Wie der abgebildete Screenshot erkennen lässt, kümmert sich die App um Anruflisten, Favoriten, SMS/MMS, sowie Lesezeichen und Browserverlauf. Die Kontakt-Sicherung haben wir bereits in der Kontakte-App integriert – somit würden hier nur noch die Kalenderdaten zu einem vollständigen PIM-Backup fehlen.

Kalender, Kontakte, SMS und Anruflisten hingegen sichert [Mobile Backup II](#) auf die Karte (hier würden also Favoriten und Lesezeichen leer ausgehen). Bei den angeforderten Berechtigungen (u. a. Internet, Telefonnummern anrufen) würde ich diese App ohne passenden Schutz (beispielsweise durch [LBE](#)) lieber nicht installieren.



Doch auch für Einzelaufgaben stehen Apps bereit. So kümmert sich beispielsweise [SMS Backup & Restore](#) (linkes Bild) um – der Name sagt es schon – die Sicherung und Wiederherstellung der Kurznachrichten. Zwar unterstützt die App (noch) keine MMS – um SMS kümmert sie sich jedoch vorzüglich, wie zahlreiche Playstore-Kommentare und Bewertungen (4,7 Sterne bei über 13.000 Bewertungen für die Gratis-Version, 4,9 Sterne bei ca. 50 Bewertungen für die werbefreie Donation-App zu knapp anderthalb Euro) belegen.

Erstellte Sicherungen lassen sich auch am Androiden sichten. Da für den Export das XML-Format gewählt wurde, steht auch einer Weiterverarbeitung am PC nichts im Wege. Vor der Erstellung eines Backups lässt sich auswählen, ob man alles oder nur ausgewählte Konversationen sichern möchte. Auch ein Versand per Mail (hier wird die erzeugte Backup-Datei an die gewählte Mail-App übergeben) ist möglich.

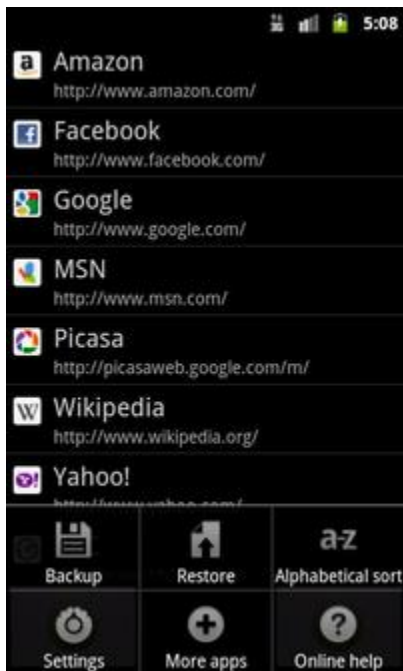
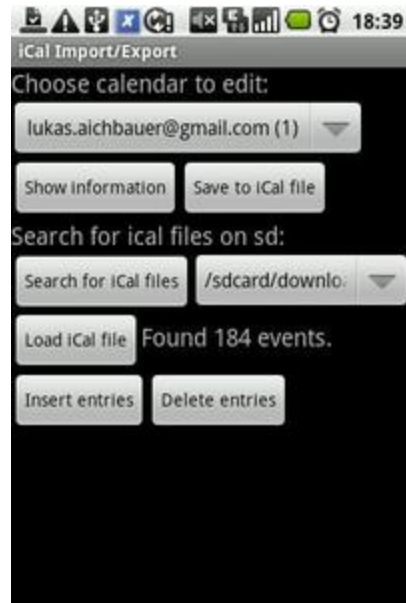
Die Internet-Berechtigung wird nur in der Gratis-Version (für Werbung) benötigt. Durch Zahlung eines Lösegelds in Höhe von knapp anderthalb Euro erhält man eine von der Werbung befreite Vollversion, die diese Berechtigung auch nicht mehr anfordert.

Tipp am Rande: Einige Kommentare erwähnen explizit den mit dieser App erfolgreich vollzogenen "Umzug" der Kurznachrichten zu einem neuen Telefon, sowie die Wiederherstellung nach einer Neuinstallation...

Wer zusätzlich noch MMS-Unterstützung benötigt, kann u. a. für knapp drei Euro auf [Textract MM & SMS Backup](#) zurückgreifen. Hier erhält man ein gut sortiertes Archiv (XML/HTML), welches sich u. a. prima am PC einsehen lässt – also eine gute Sache für die (permanente) "Langzeit-Archivierung".

Für die Anrufprotokolle käme [Call Logs Backup & Restore](#) in Frage. Der Name klingt so nach...? Ja, genau: Gleicher Entwickler wie beim gerade genannten *SMS Backup*. Und auch hier wieder Topp Bewertung (4,7 Sterne). Sieht von den Screenshots her auch wieder genau so aus – und ersetzt man "SMS" durch "Anrufprotokoll", gleichen sich auch die Features weitgehend. Größter Unterschied: Hier gibt es keine Bezahlversion, mit der man die "Internet-Berechtigung" loswerden könnte. Was übrigens auch für das aus dem gleichen Hause stammende [APN Backup & Restore](#) gilt, welches sich um die APNs (Zugangspunkte zum Internet) kümmert...

Womit wir bei den Kalendern angekommen wären, die ja – im Gegensatz zum Adressbuch – von Haus aus keine Export-Funktion (außer dem Google-Sync) haben. Hier schafft die App [iCal Import/Export](#) (rechtes Bild) Abhilfe, indem sie ihn im iCal-Format exportiert. Natürlich ist auch ein Import aus diesem Format möglich, sogar über das Web: So lassen sich beliebige andere Kalender (z. B. Feiertage) importieren. Empfehlung: Sollte das jemand für Feiertage nutzen wollen, besser einen separaten Kalender dafür anlegen – dann wird man ihn später leichter wieder los... Alternativen für ein lokales Kalender-Backup sind mir leider nicht bekannt.



Die integrierte Lösung für die Kontakte habe ich bereits im ersten Satz genannt (die alternativen Apps können da auch nicht mehr) – wenden wir uns also zu guter Letzt noch den Lesezeichen zu. Für eine einfache Sicherung und Wiederherstellung steht hier [Backup Bookmarks](#) zur Verfügung, welches die Daten im [XML-Format](#) speichert. Da es sich dabei um ein textbasiertes Datenformat handelt, lässt sich das Backup auch mit einem Blick in die Datei überprüfen – und sollte die Wiederherstellung einmal fehlschlagen, sind die Daten damit noch lange nicht futsch: Datei im Browser öffnen, jeden Link einzeln anklicken und erneut als Lesezeichen hinzufügen funktioniert sodann als Notlösung...

Oder man greift zu [Bookmark Sort & Backup](#) (linkes Bild), und holt sich damit gleich eine Sortierfunktion hinzu. Letztere lässt sich im "Automatik-Modus" betreiben, um alle Lesezeichen alphabetisch zu sortieren – oder auch manuell, wobei man Lesezeichen einzeln in ihrer Position

verschieben kann.

Für die Sicherung (und Wiederherstellung) von Lesezeichen verwendet diese App interessanterweise das HTML-Format, welches auch bei Firefox (oder dem Internet-Explorer) zum Einsatz kommt. Hübscher Nebeneffekt: Wer noch gar keine Lesezeichen angelegt hat, kann auf diese Weise die aus dem PC-Browser übernehmen...

Die Anwender sind, soweit man das den Playstore-Kommentaren entnehmen kann, durchweg zufrieden bis begeistert. Auch wenn hie und da berichtet wird, dass die Sortierfunktion mit Gingerbread (Android 2.3.x) ihre Probleme hat: Backup und Restore funktionieren einwandfrei! Warum die App allerdings "nach dem Booten starten" möchte, ist nicht so ganz nachvollziehbar. Die Internet-Berechtigung dient dem Werbebezug, und kann gegen einen knappen Euro mit der Vollversion "beseitigt" werden.

Backup in die Cloud

Das ist ja der letzte Schrei und furchtbar modern: Alles ab in die Wolke. Schon Reinhard Mey wusste (und weiß): "Über den Wolken muss die Freiheit wohl grenzenlos sein..." Da können wir unsere Backups ja gleich einmal hinterher schieben (um uns später zu wundern, wo sie teilweise wieder herab regnen).

Bordmittel

Für die Wolke gibt es sie tatsächlich – jedoch sind sie extrem eingeschränkt, und oftmals auch recht unzuverlässig. So finden sich in der Gerätekonfiguration unter [Datenschutz](#) die Optionen *Meine Einstellungen sichern* sowie *Autom. Wiederherstellung*, die sich beide auf das *Google Backup* beziehen. Details dazu sind in diesem Buch bei der entsprechenden Einstellung ausgeführt. Hier nur soviel: Es werden nur einige Apps sowie deren Einstellungen unterstützt, sowie einige Android-Einstellungen (etwa Lesezeichen und WLAN-Netzwerke). Daher an dieser Stelle lediglich die "Erwähnung der Vollständigkeit halber".

Komplettsicherungen

Sprite Backup wurde ja bereits bei den [lokalen Komplettsicherungen](#) genannt, kann aber auch in die Cloud sichern. Auf eine ganz andere Weise erledigt dies [DroiDrive](#) – indem es das ganze Dateisystem auf den Server des Anbieters kopiert. Jedenfalls soweit man auf selbiges Zugriff hat.

Um so ziemlich alles (außer Apps und deren Einstellungen) kümmert sich hingegen [miQ](#) (rechtes Bild): Kontakte, Texte, Kalendereinträge, Anruflisten, Fotos, Videos und Voicemails schiebt es auf den/die Server des Anbieters. So ganz nebenbei möchte es dabei auch "nach dem Booten starten" (automatische Synchronisation, siehe Screenshot) sowie SMS empfangen und auch senden (huch? Wozu das?). Ähnliches leistet auch [Pleex Mobile Backup](#) – mit Ausnahme des "nach dem Booten Startens" sowie der damit verbundenen automatischen Synchronisierung (aber inklusive des SMS Verschickens).



Da gibt es noch eine ganze Reihe weiterer Kandidaten, die diese Aufgabe mehr oder weniger vollständig erfüllen wollen – zu finden in der bereits bei den [lokalen Backups](#) genannten Übersicht. So richtig überzeugen konnte mich keine dieser Lösungen – was aber nicht zuletzt auch an meiner diesbezüglich etwas konservativen Einstellung liegen mag...

Kontakte, SMS & Co.

Kontakte und Kalender? Na, diese Frage beantwortet sich bei Android eigentlich fast von selbst: Wenn man dem nicht explizit einen Riegel vorschiebt, landen diese Daten automatisch auf den Google-Servern – und werden bei bestehender Datenverbindung permanent synchron gehalten (außer, wenn der entsprechende Google-Dienst mal wieder verrückt spielt). Vergleichbare Alternativen, abgesehen von den obigen, sind mir nur mit Providerbindung in Übersee bekannt. Sofern man sich nicht selbst etwas gebaut hat – doch dazu [später](#) mehr.



Für SMS und MMS scheint es jedoch mit [SMS Backup +](#) einen Kandidaten zu geben, der durchaus einen näheren Blick wert sein könnte: Von über 11.000 Anwendern mit durchschnittlich 4,6 Sternen bewertet, das will schon etwas heißen! Neben den genannten Kurz- und Multimedia-Nachrichten kümmert sich die App auch noch um die Anruflisten, und speichert alles (bei Nutzung unterschiedlicher Labels) im eigenen Google Mail-Account bzw. Kalender. Eine Wiederherstellung ist, mit Ausnahme von MMS, ebenfalls möglich. Übrigens möchte auch diese App einmal wieder nach dem Booten starten – mit einer automatischen Synchronisation dürfte also zu rechnen sein.

Nebeneffekt: Wenn SMS, MMS und Anruflisten automatisch im Google Mail-Account bzw. Kalender landen, kann man diese natürlich auch von jedem beliebigen PC ohne Zugriff auf das Telefon lesen.

Also Vorsicht, wem man die Zugangsdaten gibt – denn diese Person sieht dann sehr wohl, wann man mit wem telefoniert hat...

Wer keine MMS zu verwalten hat, und die Anrufliste ignorieren kann, der mag vielleicht alternativ zu [Gschickt](#) greifen. Neben der Online-Sicherung kommen bei dieser App einige Zusatz-Funktionen dazu, wie etwa die Nutzung von (oft kostengünstigeren) Online-SMS-Services (genannt sind hier "sms-kaufen" und "innosend"), Statistiken über den SMS-Versand, ein SMS-Editor mit Smileys, Zeichenspar-Funktion, Sprechblasen sowie anpassbarer Schriftgröße, Empfangsberichte, Text-to-Speech (TTS), und Nachrichten-Threads.

Die erweiterte Vollversion schlägt mit knapp zwei Euro zu Buche. Diese bietet unter anderem Unterstützung für "lange" SMS mit bis zu 800 Zeichen, Textbausteine, sowie mehrere Identitäten (Absender-Nummern).

Für die zusätzlichen Web-SMS-Dienste sind natürlich separate Accounts notwendig, gleiches gilt in Bezug auf die Online-Sicherung für die Website von *Gschickt* selbst. Dass die App die Berechtigung zum Versenden von Kurznachrichten fordert, sieht sicherlich jeder ein. Auch der Zugriff auf das Adressbuch ist nachvollziehbar (wenn auch etwas kritisch im Zusammenhang mit der ebenfalls erklärbaren Berechtigung zum Internet-Zugriff). Nachdenklich stimmt hingegen, dass sie zusätzlich Telefonnummern anrufen möchte. Aus der Beschreibung ist dafür kein Grund zu erkennen, eine Erklärung dazu sucht man in der App-Beschreibung ebenfalls vergeblich; auch auf der Website wurde ich diesbezüglich nicht fündig.



Synchronisation

Die Grenzen zwischen einem [Online-Backup](#) und der Synchronisation sind fließend. Kalender und Kontakte werden unter Android beispielsweise automatisch mit den entsprechenden Google-Accounts synchronisiert – manch einer sieht das als Backup an. Im Prinzip kann man sagen: Die Synchronisation ist eine Form des Backups – ein Backup aber nicht zwangsweise eine Synchronisation. Darüber hinaus gibt es Synchronisation und Synchronisation: Man kann, wie soeben anhand der Google-Dienste aufgezeigt, die Daten "mit der Cloud" synchronisieren – oder aber mit einem eigenen Rechner.

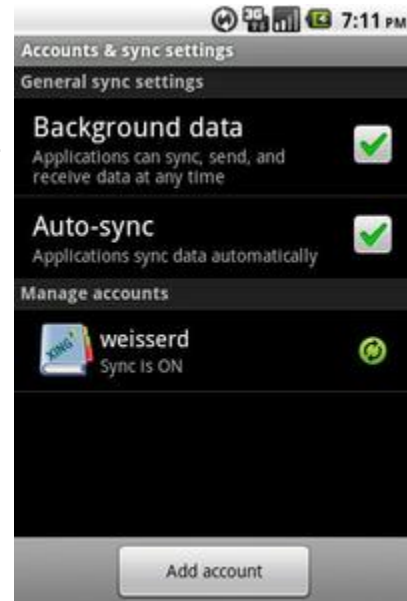
Beim Synchronisieren arbeitet man in der Regel mit zwei "gleichwertigen Kopien" (im Unterschied zum Backup, bei dem die Daten meist komprimiert abgelegt werden). Die Daten liegen also auf der "Gegenseite" genau so vor, wie sie es lokal tun. Und oftmals arbeitet man mit den Daten auch auf beiden Seiten: Fügt etwa Termine sowohl auf dem Androiden, als auch vom PC aus auf dem Google-Server hinzu.

Unterscheiden kann man unter anderem nach der Gegenseite der Synchronisation: Landen die Daten auf "fremden Servern" (also wieder in der Cloud) – oder auf einem eigenen Rechner? Ein nicht unwesentliches Kriterium,

das ich im Folgenden auch besonders berücksichtigen werde – nicht jeder gibt schließlich seine Daten freiwillig heraus.

Kontakte und die Cloud

Zuallererst wären da die beliebten sozialen Netzwerke, die sich mit in die Kontaktliste integrieren lassen. Beste App in diesem Bereich: [XING Sync](#) (Bild rechts). App ist da eigentlich das grundverkehrte Wort, denn die sucht man nach der Installation vergeblich. "Synchronisations-Adapter" trifft es eigentlich eher, oder auch "Content Provider". Wie am Screenshot zu erkennen, klinkt sich *XING Sync* nämlich in das Menü unter "Konten & Einstellungen" ein, und sorgt dort für einen zusätzlichen Eintrag – allerdings erst, wenn man dies nach Betätigen des "Hinzufügen"-Buttons veranlasst hat. Hier werden die XING Zugangsdaten hinterlegt, die Synchronisation aktiviert – und kurz danach tauchen die XING-Kontakte im Adressbuch auf. Wird einem die Liste hier zu lang, und man möchte das Ganze wieder loswerden: Keine Panik, die Daten sind nicht bunt mit den Standard-Kontakten gemischt worden. Einfach an der genannten Stelle das Konto wieder entfernen, und weg sind sie. Bitte beachten: *XING Sync* funktioniert nur für XING Premium-Accounts! Und kostet übrigens einen Euro.



Auch die [originale XING-App](#) soll die Synchronisation mittlerweile beherrschen. Ob sie das genau so gut hinbekommt, ist mir nicht bekannt – dafür ist sie jedoch gratis, und fast gleich gut bewertet.

Von XING zu den Mitbewerbern: Orkut-Kontakte lassen sich mit [Orkut Contact Sync](#) in das Adressbuch einbinden. Wer hier nach der Installation die App vergeblich sucht, fühlt sich eventuell an die Beschreibung von *XING Sync* erinnert. Und verfährt hier erfolgreich analog zu selbiger.

Ich höre schon die Frage: Warum wird hier nichts von Facebook erwähnt? Da scheint es einfach nichts vernünftiges zu geben. Was sicher nicht daran liegt, dass ich kein großer Fan von Facebook & Co. bin: Die beste Bewertung liegt hier bei 3,5 Sternen, für die entweder keinerlei Details zu sehen sind – oder die App funktioniert einfach nicht mehr mit aktuellen Android-Versionen. Tut mir leid für die danach Suchenden – aber es besteht ja immer noch die Hoffnung, dass mir da etwas entgangen ist...

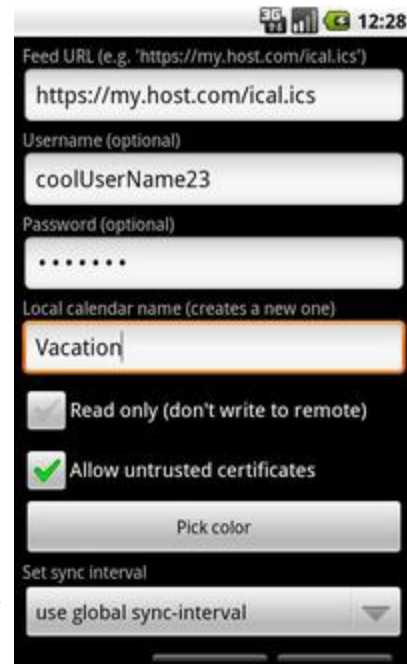
Weitere Apps zu diesem und dem nächsten Thema finden sich übrigens in [dieser Übersicht](#).

Kontakte, Kalender und mehr mit eigener Hardware synchronisieren

Für die Kalender-Synchronisation hat sich in diesem Bereich eine noch recht junge App schnell an die Spitze gesetzt: [ICSSync](#) (rechtes Bild) kann mehrere *.ics Dateien mit lokalen Kalendern synchron halten. Dafür wird für jede Quelle ein eigener lokaler Kalender angelegt. Die Quellen können dabei auf einem FTP- oder auch einem CalDav-Server liegen. Ein einseitiger Import (etwa von Feiertagen) ist dabei ebenso möglich wie eine bidirektionale Aktualisierung, bei der die Kalenderdatei gleichzeitig von mehreren Clients zum Lesen und Schreiben genutzt wird. Der Abgleich kann sowohl manuell als auch automatisiert erfolgen – das wäre also so etwas wie Google Kalender ohne Google, wenn man das Web-Interface einmal vernachlässigt.

Ist eine automatische Synchronisierung gewünscht, lässt sich bei [ICSSync](#) das Intervall für jede Quelle individuell festlegen. Eine Synchronisation abhängig vom verfügbaren Netzwerk (etwa nur bei bestehender WLAN-Verbindung) ist ebenfalls möglich.

Die verlinkte Testversion ist für 20 Tage in vollem Umfang einsatzfähig. Will man sie weiterhin nutzen, muss man für knapp anderthalb Euro einen Freischaltcode erwerben – was sowohl im Google Playstore (Zahlung per Kreditkarte) als auch bei AndroidPIT (Zahlung per Kreditkarte oder Paypal) möglich ist.



Eine ganze Menge mehr "Services" lassen sich mit [Funambol Sync](#) abgleichen, wie der Screenshot (links) zeigt: Kontakte, Kalender, Bilder, Videos und sonstige Dateien. Am liebsten sieht es der Anbieter natürlich, wenn auf der "anderen Seite" seine eigenen Server werkeln – man dazu also die Services von [Funambol.COM](#) selbst nutzt. Das ist selbstverständlich eine Möglichkeit, aber nicht die einzige. Auf [Funambol.ORG](#) stehen sowohl Server-Software für Linux und Windows (beides für 32Bit und 64Bit), als auch PC-Clients für Windows und Mac OS bereit.

Ganz unabhängig davon gibt es auf der letztgenannten Website auch Lösungen für zahlreiche PC-Programme wie Mozilla Thunderbird und Gnome Evolution, und viele weitere. So gesehen deckt [Funambol](#) wohl die breiteste Palette ab. Wer mehr dazu erfahren möchte, sollte also dort einmal vorbei schauen – und findet dann unter anderem auch einige [Video-Demos und -Tutorials](#), Online-Hilfe und Kontakt zur zugehörigen Community.

Um es noch einmal explizit kundzutun: Sowohl die Android-App als auch die zugehörigen PC-Komponenten sind gratis erhältlich. *Funambol* ist auch nicht "gänzlich unbekannt", sodass man mit Hilfe beim Einstieg und auch bei Problemen sowohl seitens der Entwickler als auch aus der Community rechnen kann. Einiges an Dokumentation (beispielsweise zum Aufsetzen und Administrieren eines *Funambol* Servers) ist auf [dieser Webseite](#) zusammengetragen.

Es gibt noch eine Reihe weiterer Lösungen – sowohl die Android-Apps als auch die Gegenstücke auf den heimischen Computern betreffend. Die zugehörige Übersicht habe ich bereits am Ende des vorigen Themas verlinkt.

Dateien und Verzeichnisse in der Cloud

Unbewusst hat Reinhard Mey die Cloud (eine Übersicht dazu wieder [im Forum](#)) bereits 1971 in seinem Lied "Über den Wolken" besungen. Hier einmal eine zeitgenössische Interpretation des Refrains:

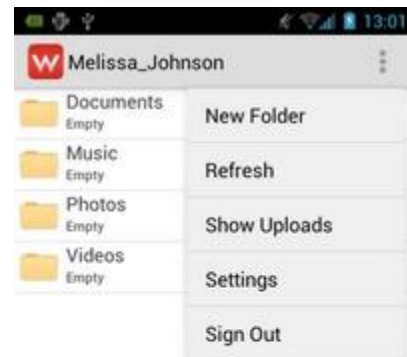
Über den Wolken *[total abgehoben und völlig überbewertet]*
Muss die Freiheit wohl grenzenlos sein. *[die Daten werden über die halbe Welt verstreut!]*

Aller Kummer, alle Sorgen, sagt man, *[genau weiß es keiner]*
Blieben darunter verborgen, und dann, *[und niemand schaut da wirklich durch]*
Würde, was uns groß und wichtig erscheint *[Megabytes, Gigabytes]*
Plötzlich nichtig und klein. *[angesichts der verfügbaren Terra- und Petabytes]*

Bei dem Wort "Cloud" denkt der Android-Jünger wahrscheinlich zuerst an Google, dann an die Dropbox. Danach folgt wahrscheinlich erst einmal eine ganze Weile – gar nichts. Dennoch möchte ich hier weder mit Google, noch mit Dropbox beginnen – sondern eine überaus interessante Alternative zu letzterer vorstellen:

Der Dienst nennt sich **Wuala**, und dahinter steht u. a. der bekannte französische Festplatten-Hersteller **LaCie**. Die Server finden sich in Frankreich, der Schweiz und Deutschland – sind also gar nicht so weit weg.

Aber das sind alles nur "politische Details". Interessanter sind einige der technischen "Kleinigkeiten": Etwa, dass dieser Cloud-Service sowohl für Android, als auch unter Linux, Mac OS und Windows Clients bereitstellt. Oder dass die Daten in der Cloud verschlüsselt abgelegt werden – wobei die Verschlüsselung auf dem Client erfolgt. Ferner lassen sich die Daten für andere freigeben, sogar Gruppen kann man anlegen: So steht dem Teamwork eigentlich nichts im Weg. Als Sahnehäubchen wurde dem Ganzen eine Versionierung spendiert: Die ältere (oder gelöschte) Version einer Datei zu "restaurieren" stellt hier also kein großes Problem dar.



Der "offizielle Android-Client" (rechts abgebildet) trägt den Namen des Dienstes: [Wuala](#). Er unterstützt diese Funktionalitäten natürlich. Ein Drittanbieter steuert noch weitere Apps dazu bei: Etwa eine [Galerie](#), einen [Music Player](#) und einen alternativen [Sync-Client](#), der allerdings nur lesend auf die Wuala-Cloud zugreifen kann.



Eine der größten und bekanntesten "Wolken" ist sicher [Dropbox](#) – mit Client-Software für Linux, Mac OS und Windows auf dem heimischen Rechner, sowie Android, BlackBerry, iPad und iPhone auf den mobilen Geräten, wird hier sicher auch die größte Bandbreite von Geräten unterstützt. Darüber hinaus gibt es für Entwickler die Möglichkeit, durch die Verwendung der bereitgestellten API die *Dropbox* direkt in ihre Applikation einzubinden – was bei vielen Apps auch bereits passiert ist. Um welche Apps es sich dabei handeln könnte, kann man im [Dropbox App-Verzeichnis](#) für alle beteiligten Plattformen (auch separat gefiltert) nachschlagen. Wobei diese Liste keinesfalls vollständig ist...

Aber schauen wir uns doch ausgewählte Clients einmal kurz an. Da wäre natürlich zuerst die [offizielle Dropbox-App](#) (siehe Bild links), die genau wie der Dienst einfach nur *Dropbox* heißt. Sie kümmert sich in erster Linie um die Synchronisation von Dateien und Verzeichnissen, wozu sie einen "Spiegel" der im Service bereitgestellten "Online-Festplatte" auf dem Gerät bereithält. Wird dort eine Datei abgelegt oder verändert, bemerkt der Service das, und veranlasst eine Online-Aktualisierung – gegebenenfalls wartet sie damit auch, bis eine Netzverbindung besteht. Umgekehrt erfolgen auch Aktualisierungen aus der Cloud, falls eine Datei von einem anderen Client modifiziert wurde. So sind alle Beteiligten stets auf dem aktuellen Stand.

Ebenso lassen sich Dropbox-Ordner für Freunde und Verwandte freigeben. Oder EMail-Anhänge direkt in der Dropbox abspeichern. Oder Dokumente direkt in der Dropbox bearbeiten... 4,7 Sterne bei weit über 70.000 Bewertungen zeugen nicht nur von reger Nutzung, sondern auch von vielen zufriedenen Anwendern.

Für die [Automatisierung mit Tasker](#) stehen gleich zwei Plugins zur Verfügung: [Dropbox Sync for Tasker Locale](#) (rechtes Bild) greift direkt auf die offizielle Dropbox-API zu, und benötigt so keine zusätzliche App. Der Name verrät bereits, dass sich dieses Plugin auch mit *Locale* verwenden lässt – das volle Potential wird aber nur mit *Tasker* ausgeschöpft, indem sich z. B. auch Variablen in Datei- und Verzeichnisnamen verwenden lassen.

Das [DropSpace Plugin for Tasker](#) greift hingegen auf die App [DropSpace](#) zurück, mit der sich Dateien und Verzeichnisse für die Synchronisation mit der Dropbox auswählen lassen. Diese werden in einer Liste zusammengefasst, und entweder "on Demand" per Widget, oder "automatisch zeitgesteuert" per Service synchronisiert.

Ein weiterer Dropbox-Client wäre in diesem Zusammenhang [DropSnap](#). Auch hier lassen sich wieder Verzeichnisse zur Synchronisation konfigurieren. Ferner gibt es einige Möglichkeiten, die Synchronisierung an die eigenen Bedürfnisse anzupassen: Nur bei vorhandener WLAN-Verbindung synchronisieren? Soll überhaupt automatisch synchronisiert werden? Fotos gleich nach dem "schießen" automatisch hochladen? Das Synchronisationsintervall lässt sich bei dieser App ebenfalls einstellen.

In Sachen Verschlüsselung sah es bislang ein wenig mau aus, wollte man dabei zu verschiedenen Systemen kompatibel sein (also etwa vom Androiden *und* vom PC unter Windows *und* Linux auf die Daten zugreifen). Besserung zeichnet sich bereits ab: [BoxCryptor](#) gibt es jetzt auch für Android. Die noch recht frühe Version unterstützt zwar derzeit ausschließlich lesenden Zugriff auf das mit [EncFS](#) verschlüsselte Verzeichnis, doch mehr Funktionalität dürfte sicher bald folgen.





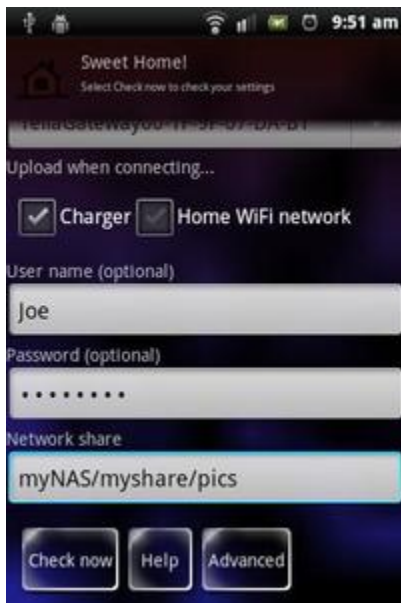
Bei der Suche nach weiteren Diensten gestaltet sich das Finden brauchbarer Antworten recht schwierig: Wer vertraut beispielsweise einem koreanischen Dienstleister – selbst wenn (oder gerade weil?) dieser ihm gleich 50GB an Speicherplatz zur Verfügung stellt – die App dafür aber gleich mit den *Kontaktdaten* ins Internet möchte ([Daum Cloud](#))? Noch dazu, wenn die App keinen einzigen "lesbaren Buchstaben" enthält. Derart vorgewarnt, stutzt man über gleichartige Berechtigungen auch bei kalifornischen Anbietern ([SugarSync](#)), obwohl das "Storage" hier auf 5GB limitiert ist. Bei einem weiteren Kandidaten ([Syncbox](#)) reicht dann bereits die Kombination aus 15GB Storage und chinesischem Anbieter, dass man die Stirn in Falten legt – obwohl die App selbst nicht schlecht aussieht, auch was die geforderten Permissions betrifft. Erst einmal abwarten, was andere dazu schreiben...

Einen Kandidaten möchte ich dennoch an dieser Stelle benennen: [DCD CloudSync](#) (linkes Bild; man beachte den Tornado zwischen den oberen beiden Buttons, der hoffentlich nicht in Bezug auf die Daten bedeutet: "Up, up – and away..."). Ganz frisch auf dem Markt, kümmert diese App sich um Dateien und Verzeichnisse in der Cloud von Google. Wer einen Google-Account hat (also nahezu jeder Android-Nutzer), kann sich hier Speicher freischalten lassen. Voraussetzung ist allerdings, dass eine Kreditkarte mit dem Account verknüpft ist – auch wenn der Service selbst gegenwärtig kostenlos sein soll. In der Gratis-Version kann man mit *DCD CloudSync* genau einen Ordner verwalten; die [Pro-Version](#) für knapp 3 Euro enthält dieses Limit nicht. Eine Synchronisation kann automatisch bei jeder Änderung, in vorgegebenen Zeitintervallen, oder auch manuell erfolgen.

Dateien und Verzeichnisse auf eigener Hardware

Zugegeben: Nicht jeder kann sich gleich einen fetten Serverpark in den Keller stellen. Aber hey, auch MicroSoft hat einmal ganz klein in einer Garage angefangen. Und wenn wir gerade vom Windows-Hersteller sprechen: Den kann man doch auch freigeben. Ich meine, den Speicherplatz auf dem Windows-Rechner. Auf Linux und Mac OS Rechnern heißt es dazu: "Tanze **Samba** mit mir...", denn hier gibt es den freien **Samba**-Server – bei den meisten Linuxen von Haus aus im Repository enthalten, und auch für Mac OS zu haben. Auch **NAS**-Systeme bieten dies an, verbrauchen im Vergleich zum herkömmlichen PC kaum Strom – und sind mittlerweile auch recht erschwinglich. So lässt sich Speicherplatz im heimischen Netzwerk zur Verfügung stellen – und darauf können dann auch unsere Androiden zugreifen.

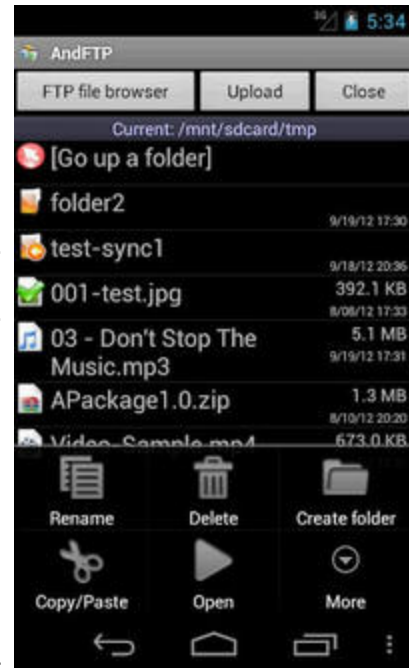
Beispielsweise mittels **AndSMB** (rechtes Bild), einem SMB-Client für Android. Diese App erlaubt Zugriffe auf Windows-Freigaben ebenso wie auf Samba-Freigaben eines Linux- oder Mac OS-Rechners, handelt es sich doch um das gleiche Übertragungs-Protokoll. Ein Dateibrowser ist in der App enthalten, und erlaubt sowohl den Up- als auch den Download. Automatisiert man das Ganze nun, was mit dieser App möglich ist, hat man eine Synchronisation von Ordnern – und somit eine eigene minimale Cloud.



Alternativ könnte **Sweet Home** (linkes Bild) zum Einsatz kommen. Die Beschreibung der **Pro-Version** dieser App bringt es auf den Punkt: Keine Kabelsuche, auch muss man den Androiden nicht mehr umständlich zerlegen, um die SD-Karte zu entnehmen: Sobald man zu Hause ankommt, und sich der Androide ins heimische WLAN verbindet, werden die Daten automatisch synchronisiert. Die Gratis-Version punktet (laut Beschreibung) mit häufigen Updates (sowie dem unvermeidlichen "Nagging"), während die Bezahlversion gut getestet und stabil ist. Benutzer der freien Version sind also Beta-Tester...

Da Samba bzw. Windows-Freigaben in der Regel jedoch auf das jeweilige lokale Netzwerk beschränkt sind, ist man mit einer derartigen Mini-Wolke örtlich jedoch recht limitiert. Für viele Fälle mag dies ausreichend sein – und was nicht über fremde Netze geht, ist auch vor "Schnüfflern in der Mitte" sicherer. Wird jedoch ein Zugriff auch von unterwegs gewünscht, greift man am Besten zu einem der folgenden Multi-Talente, die sich auch auf Protokolle wie **FTP[S]** und **SFTP/SSH** verstehen. Und auf der Gegenseite natürlich zu einem passenden Server: SFTP/SSH sind unter Linux von Haus aus dabei, ein FTP-Server lässt sich auf allen Systemen relativ einfach aufsetzen – und verfügt man über einen eigenen Root-Server für z. B. seine Webseiten, sollte einer der beiden Dienste (SSH-Server und/oder FTP-Server) ohnehin bereits vorhanden sein.

Eine passende App auf unserem Androiden wäre dann beispielsweise [AndFTP](#) (rechtes Bild). Topp bewertet, bringt diese gar einen Manager für die auf dem Server abgelegten Dateien mit. Sie versteht sich auf die Protokolle FTP, FTPS, SCP sowie SFTP, und kann mehrere Serverkonfigurationen verwalten. Neben Upload, Download und Synchronisation ist auch das Share-Menü sowie direkte Unterstützung für die Galerie eingebunden. Für SSH Verbindungen (SCP) können überdies RSA/DSA Schlüssel verwendet werden. SCP und Ordner-Synchronisation sind allerdings nur in der Pro-Version verfügbar.



Wer es jedoch besonders flexibel mag, auf einen "Remote Explorer" verzichten kann, und hauptsächlich Wert auf die Synchronisation legt – der sollte unbedingt einen Blick auf [FTPSyncX](#) (Bild links) werfen, denn dies ist mit ziemlicher Sicherheit die erste Wahl: Wie beim gerade beschriebenen *AndFTP* werden auch hier die Protokolle FTP, FTPS sowie SFTP/SCP unterstützt, zusätzlich ist Samba mit an Bord. Doch bereits die Testversion ist im Funktionsumfang völlig uneingeschränkt – sieht man vom Limit "2 Server mit insgesamt 4 Verzeichnissen" einmal ab, was für einen Test (und oftmals auch für den Alltags-Einsatz) völlig ausreichend ist.

Okay – und was ist der Funktionsumfang? Synchronisieren, oder? Klar, aber wie? Und da stehen hier wirklich alle Türen offen: Einzelne Verzeichnisse, oder gleich alle für einen Server definierten Verzeichnisse synchronisieren? Zeitgesteuert alle XX Minuten, oder nur manuell? Und wenn manuell: Aus der App heraus, über

Widgets, oder per Shortcut? Alles ohne großen Aufwand möglich. Die gewünschten Intervalle lassen sich auch für jedes Verzeichnis und jeden Server separat und unabhängig definieren.

Naja, aber dann läuft die App also ständig im Hintergrund? Mir wäre es lieber, sie synchronisiert einmal, wenn... Ein Fall für [Tasker](#), eindeutig. Und auch für diesen ist ein Plugin bereits im Funktionsumfang – es sollte also kein Thema sein, die Synchronisation z. B. einmal automatisch anzustoßen, wenn man daheim ankommt und sich mit dem heimischen WLAN verbindet. Oder wenn man den Androiden in die Docking-Station steckt. Oder... Der Fantasie sind da keine Grenzen gesetzt.

Wer mehr Server und/oder Verzeichnisse zu synchronisieren hat: Für knapp anderthalb Euro gibt es die [Pro-Version](#). Für das, was die App bietet, absolut erschwinglich.

Zumindest erwähnen sollte ich auch noch die App [FolderSync](#), bei der an unterstützten Protokollen neben FTP, FTPS, und SFTP noch WebDAV und Samba dazukommen. Und soll es doch einmal in die Wolke gehen, sind offensichtlich Dropbox und Amazons S3 ebenso mit von der Partie. Außerdem ist noch ein Dateimanager integriert, mit der sich auch die Remote gespeicherten Daten verwalten lassen.

Vergleich von Synchronisations-Services in der Cloud

Mit freundlicher Genehmigung des [Linux-User Magazins](#) darf ich an dieser Stelle die in Ausgabe 09/2011 (Artikel "[Datenwolke](#)", Seite 25) dargestellte Übersicht verwenden. Eine Spalte (Teamdrive) habe ich dabei ausgelassen, da hier (zumindest derzeit) kein Android-Client existiert.

	Ubuntu One	Dropbox	Spideroak	Wuala	Zu
URL	one.ubuntu.com	www.dropbox.com	spideroak.com	www.wuala.com	www.zu.com
Gratis-Speicher	5 GByte	2 GByte	2 GByte	1 GByte	
Linux-Client	ja	ja	ja	ja	
Mac-Client	nein	ja	ja	ja	
Windows-Client	in Entwicklung	ja	ja	ja	
Mobil-Client	Android, iPhone	Android, iPhone, BlackBerry	Android, Maemo, iPhone	Android, iPhone	Android
Sync / Backup	ja / nein	ja / ja	ja / ja	ja / ja	
Web-GUI / WebDAV	ja / ja	ja / nein	ja / nein	ja / nein	
Sharing / Multi-User	ja / nein	ja / nein	ja / ja	ja / nein	
Lokalisierung	ja	ja	nein	ja	
Vorteile	Client in Ubuntu bereits enthalten, Integration mit anderen Canonical-Cloud-Diensten	großer Funktionsumfang, weit verbreitet, viele Clients, externe Zusatztools,	Hervorragendes Sicherheitskonzept, leistungsfähiger Linux-Client, Clients für Android und Maemo	ungewöhnliches Konzept mit verteilter Speicherung, kostenlose und flexible	Online Netzwerk

		Webinterface, preiswert bei kostenpflichtigen Versionen		Speichererweiterung mit Option <i>Festplatte tauschen</i> möglich, vorbildlicher Client	
Nachteile	durchschnittlicher Funktionsumfang, teuer	relativ umständlich	nur in Englisch verfügbar	hohe Systemlast	Funk

(Quelle: [Linux-User](#))

Security

Wie [eingangs](#) erwähnt, soll es nun um den Schutz vor Fremdzugriffen gehen. Diesbezüglich möchte ich hier mehrere Ebenen unterscheiden:

- [Verschlüsselung](#): Besonders interessant, wenn Daten auf fremden Servern (Stichwort: Cloud) abgelegt werden. Da es auf selbigen nicht mehr in der Macht des Daten-Eigentümers liegt, den "physikalischen Zugriff" abzusichern, sichert man auf diese Weise "logischen Zugriff" ab: Selbst wenn ein "Angreifer" an die Dateien gelangt, muss er damit noch lange nicht an die Daten (Inhalte) kommen.
- [Zugriffsrechte](#): Worauf darf eine App zugreifen? Unschön, wenn da jemand im Hintergrund ungefragt und ungewünscht sämtliche Kontaktdaten "irgendwo hin" verschickt. Oder andere böse Sachen mit unseren Daten anstellt.
- [Malware und Viren](#): Sollten eigentlich mit dem vorigen Punkt bereits abgearbeitet sein, werden aber dennoch explizit separat behandelt.
- [Location-Cache](#): Noch ein "Datenschnüffler", mit dem sich Bewegungsprofile erstellen ließen ("Herr X verlässt werktags fast jeden Vormittag zwischen sieben und acht die Wohnung, und geht zu Y (vermutlich Arbeit). Diesen Ort verlässt er dann am späten Nachmittag, und geht zweimal die Woche in den P... (Pizza-Laden natürlich)..."). Solches soll zwar laut offizieller Verlautbarung nicht passieren – aber wie sieht es inoffiziell... oder so?

Wer sich genauer informieren möchte, welche Schutzmaßnahmen Android selbst anwendet, findet in der Computerwoche [einen interessanten Artikel](#), der sich dem Thema eingehend widmet.

Verschlüsselung

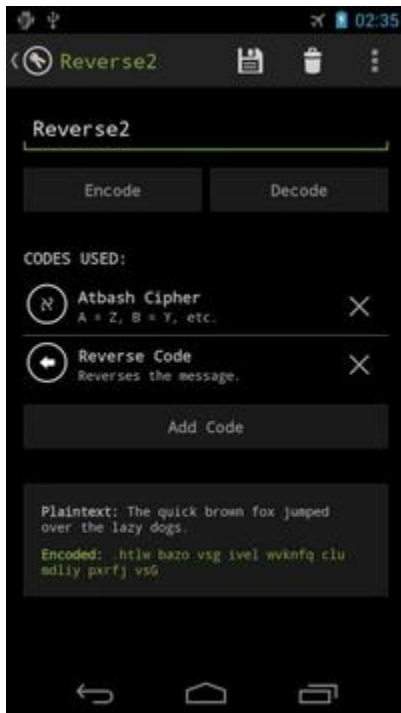
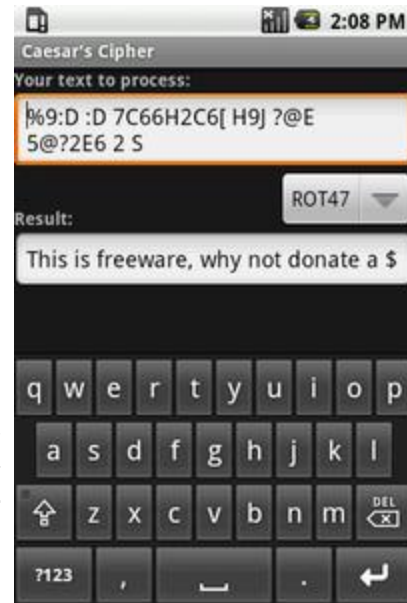
Texte Verschlüsseln

(Eine Übersicht mit Apps zum Thema Textverschlüsselung findet sich [im Forum](#).)

Das tat schon Gaius Julius Caesar – weshalb einer der einfachsten Codes auch nach ihm benannt wurde: die [Caesar-Verschlüsselung](#). Die bekannteste Variante aus dieser Reihe trägt auch den Namen "ROT-13" – was nicht etwa darauf hinweisen soll, dass Caesar damit die Cleopatra 13 Mal zum Erröten brachte (tat er bestimmt nicht: Nach Sueton verwendete Caesar nämlich ROT-4). Vielmehr heißt es: ROTiere jeden Buchstaben 13 Mal – aus "A" wird also "N". So wird aus dem Wort "Text" das etwas schwerer lesbare "Grkg". Zur Entzifferung verschiebt man die Buchstaben einfach zurück (aus "N" wird "A").

Von einer sicheren Kodierung kann hier also beileibe nicht die Rede sein – doch veranschaulicht dies wunderbar, wie Verschlüsselung prinzipiell funktioniert. Genutzt wird es trotzdem – weniger zu Sicherheitszwecken, als vielmehr um z. B. unbeabsichtigtes Lesen von Spoilern zu vermeiden. Zumindest Geo-Cachern dürfte diese Verschlüsselungsart auch vertraut vorkommen...

So wird die App [Caesar's Cipher](#) (Bild rechts) von ihren Benutzern als als hilfreiches und nützliches Tool beim Geo-Caching beschrieben (hier werden Hints kodiert und dekodiert). Oder, wie es ein englischer Kommentar formuliert: *Very useful. Now my teacher has now idea what the gibberish on my paper says.* Hoffentlich war da nicht der Geschichtslehrer gemeint...



Caesars Code hat natürlich einen großen Haken: War er zu Zeiten des genannten Römers noch relativ sicher, benötigt man heutzutage für das "Knacken" des Codes wahrscheinlich weniger als eine Sekunde – bedenkt man, dass ein Computer lediglich 26 (bei reinen Großbuchstaben) bzw. maximal 256 Varianten (8-Bit ASCII) dafür durchspielen muss. So schützt ein reiner Caesar-Code allenfalls vor dem "schnellen Lesen", nicht aber vor dem Lesen an und für sich.

Doch was galt bereits im Altertum als besonders kompliziert? Da war doch was mit einem [Gordischen Knoten](#), an dessen Auflösung sich viele schlaue und starke Männer vergeblich versuchten. Nach diesem ist offenbar die App [Gordian Secret Code Tool](#) (Bild links) benannt. Diese nutzt prinzipiell ähnliche Methoden – erlaubt jedoch, sie zu kombinieren. Zugegeben: Ein Caesar-4 mit anschließendem Caesar-9 ergäbe auch nur wieder ROT13. Kombiniert man ihn jedoch beispielsweise mit Vigenère, wird die

Sache schon komplexer. Ob der potentielle "Knacki" deswegen gleich ins Schwitzen kommt, steht natürlich auf einem anderen Blatt...

Bleiben wir zunächst noch bei den Legenden – machen jedoch einen größeren Zeitsprung über ein paar Jahrtausende. Damals war es vielen ein Rätsel, was das deutsche Militär da so funkte. Auf Griechisch heißt Rätsel "αίνιγμα", mit lateinischen Buchstaben schreibt man es [Enigma](#) – die Rede ist natürlich von der bekannten Rotor-Schlüsselmachine. Dieses Original benutzte auch eine Kombination, und zwar waren hier mehrere Walzen hintereinander geschaltet, die nach jedem Buchstaben auch noch wie ein Kilometerzähler weiterbewegt wurden. Dies bedeutet, dass sich der Schlüssel nach jedem Buchstaben ändert: Hätte Caesar aus einem *OTTO* vielleicht ein *GLLG* gemacht, wäre bei einer Enigma eher etwas wie *PQWS* dabei herausgekommen. Daran hatten die Alliierten im zweiten Weltkrieg eine ganze Weile zu knabbern.

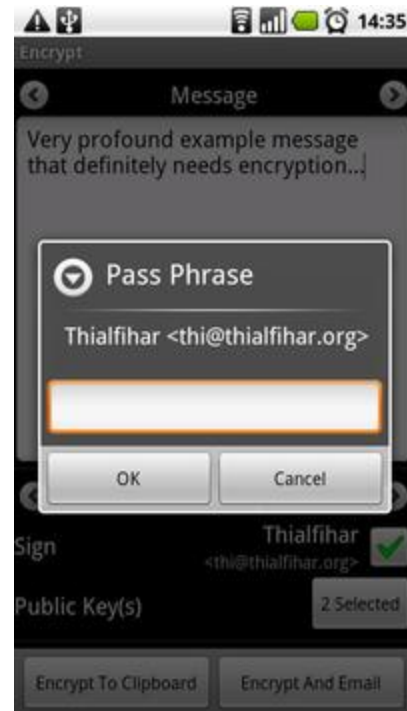
Wer dieses Feeling einmal nachempfinden möchte, bekommt mit der App [Pocket Enigma Machine](#) Gelegenheit dazu: Der Entwickler hat hier Bild- und Tonaufnahmen des Originals verwendet, damit sich das Ganze möglichst "echt" anfühlt.



Nah dran waren wir ja nun bereits, doch nun wollen wir uns endgültig der Gegenwart zuwenden. Spricht man heute von einer "sicheren Verschlüsselung", so fallen Begriffe wie [RSA](#) (asymmetrisch) oder [DES](#) (symmetrisch). Den Nachfolger des letztgenannten ([AES](#)) macht sich [Crypt Haze](#) (Bild links) zu Nutze, um Mails und Kurznachrichten zu ver- und entschlüsseln (wozu es natürlich die entsprechenden Zugriffs-Rechte auf Kontakte und SMS anfordert – allerdings auf die allgegenwärtige Internet-Berechtigung verzichtet). Auch die App [Encrypt It](#) setzt auf eine 256Bit AES Verschlüsselung, und verbindet diese zur Erhöhung der Sicherheit noch mit einem [Salt](#). Diese App verlangt keinerlei Permissions.

Was aber verwendet der sicherheitsbewusste Mensch von Heute, wenn er Nachrichten verschlüsselt oder signiert übertragen möchte? Das Zauberwort heißt **GPG**. Das "G" steht hier für "GNU", und "PG" für den "Privacy Guard" – also den Beschützer der Privatsphäre. Hier werden Schlüsselpaare verwendet, bestehend aus einem "privaten" und einem "öffentlichen" Schlüssel. Die Namen sagen es bereits: Den einen hütet man wie den eigenen Augapfel, den anderen gibt man frei heraus. Dank dieses öffentlichen Schlüssels ist nun jeder Anwender von GPG in der Lage, eine Nachricht so zu verschlüsseln, dass sie nur vom Besitzer des zugehörigen privaten Schlüssels wieder dekodiert werden kann. Umgekehrt lassen sich mit dem privaten Schlüssel Nachrichten signieren. Die Echtheit dieser Signatur kann jeder mit dem frei verfügbaren öffentlichen Schlüssel überprüfen.

Soweit die Theorie und das grobe Prinzip, kommen wir zur Praxis. Die bekannteste App in diesem Segment ist sicherlich **APG** (Bild rechts). Als Open-Source Software stellt das Projekt sicher, dass es keine "Hintertüren" geben kann (die würden schnell entdeckt). Durch die Verwendung von Schlüsselpaaren (wie soeben beschrieben) wird darüber hinaus auch eine relativ hohe Sicherheit gewährleistet – zumindest, so lange der private Schlüssel nicht kompromittiert wurde.



Mit der App lassen sich PGP/GPG Schlüsselpaare verwalten sowie Mails und Dateien verschlüsseln, signieren, entschlüsseln und die Signaturen überprüfen. Unterstützt wird Gmail, aber auch in K9 lässt sich die App integrieren; für andere Apps wird eine API bereitgestellt. Seit neuestem werden auch Schlüsselservers unterstützt (daher auch die Internet-Berechtigung). 4,5 Sterne bei über 900 Bewertungen bestätigen die Qualität dieser App aus Anwendersicht.

Alternativ bietet sich noch der **OpenPGP Manager** (Bild links) an, dessen Funktionsumfang annähernd dem von APG entspricht – wenn man die K-9 Integration sowie die API abzieht. Auch hier ist eine vollständige Schlüsselverwaltung (wie abgebildet) enthalten, einschließlich der Unterstützung von Keyservern.

Passwort-Safes

Ein ganz spezieller Fall von "kurzen Texten", die es um jeden Preis abzusichern gilt, stellen sicher Passwörter und PIN-Codes dar – eine passende Übersicht verfügbarer Apps [findet sich hier](#). Eine App, der ich solche Daten anvertraue, sollte möglichst mein Handy auch nicht verlassen – sonst bestünde ja immer die Möglichkeit, dass sie diese sensiblen Daten (von mir unbemerkt) an Dritte weiterreicht. Zum Glück sehen das viele Entwickler ähnlich – anderen ist hingegen beispielsweise wichtiger, auch ein "Backup" in der Cloud ablegen zu können. Das wäre aus meiner Sicht allerdings, wenn überhaupt, Aufgabe einer anderen App aus anderem Hause – ich bin halt ein wenig paranoid. Daher werde ich mich hier auf Apps beschränken, die über keine eigene "Internet-Berechtigung" verfügen.

Da wären zum Beispiel die PINs von EC- und Kreditkarten. Der Eine oder die Andere kennt vielleicht noch diese kleinen "Geheimkärtchen", auf denen man sich diese in der Regel vierstelligen Zahlencodes "verschlüsselt" notieren kann – und genau dieses Prinzip greift [PIN'r](#) (rechtes Bild) auf: Die PINs werden in einer Tabelle voller Zufallszahlen versteckt. Nur wer das Geheimwort kennt, findet die richtige Kombination wieder. Dumm für den Schnüffler: Jedes beliebige "Geheimwort" liefert passende Zahlencodes, mit "Passwort erraten" ist es also auch nicht ganz so einfach...



Doch oftmals geht es um mehr als nur PINs: Zugangsdaten für Foren und Websites, Kreditkartendaten, Mailkonten, Lizenzschlüssel für Software... Zahlreiche Daten sind sensitiv genug für einen Passwort-Safe. Da bieten sich Apps wie beispielsweise [Password Safe](#) (linkes Bild) an. Für jeden Anwendungsbereich lassen sich hier Vorlagen (Templates) definieren: Für ein Mailkonto benötigt man in der Regel drei bis vier Felder (Name, Mail-Adresse, Passwort sowie ggf. zugehörige Webseite) – für weitere Mailkonten sind es die gleichen vier Felder. Wie es im Fall von Kreditkarten aussehen könnte, zeigt der linke Screenshot. Einige Templates bringt die App gleich mit, weitere kann man selbst erstellen (bzw. die vorhandenen entsprechend anpassen). Jeder Vorlage lässt sich darüber hinaus auch ein Icon zuordnen, was die Erkennung vereinfacht. Bei Erfassung der Daten wählt man dann einfach das jeweilige Formular (also die Vorlage) aus, und füllt die Daten in die Felder; selbst jetzt lassen sich bei Bedarf noch zusätzliche Felder hinzufügen.

Da so eine Datensammlung schon einmal etwas größer ausfallen kann, ermöglicht *Password Safe* auch eine Einteilung in Kategorien; die Darstellung erfolgt hier in einer Baumstruktur, in der man sich leicht zurechtfinden kann. Davon kann man sich in einer Gratis-Version der App zunächst überzeugen – die erfassten Daten lassen sich in die für ca. zwei Euro erhältliche Vollversion übernehmen, die dann auch einen Export/Import im CSV-Format ermöglicht.

Und bevor ich es zu erwähnen vergesse: Damit die Daten vor fremden Blicken geschützt sind, werden sie mit 128Bit [AES](#) verschlüsselt gespeichert. Wem das noch nicht ausreicht, der kann einen Blick auf [Passman](#) werfen. Hier gibt es zusätzlich eine "Selbstzerstörung": Nach einer konfigurierbaren Anzahl an Fehlversuchen bei der Eingabe des Master-Passwortes werden sämtliche erfassten Daten automatisch gelöscht. Ein etwaiger Handy-Dieb hat somit wenig Chancen – und man selbst hat ja hoffentlich ein Backup der Datenbank an anderer Stelle (z. B. auf dem PC) abgelegt.

Natürlich gibt es noch jede Menge [weiterer Apps zu diesem Thema](#). Solche, die den beschriebenen ähneln – aber auch andere, die aber zumeist die Berechtigung für den Internet-Zugriff (meist für eine Speicherung der verschlüsselten Datenbank bei Dropbox & Co) einfordern.

Dateien und Verzeichnisse

Da fällt einem sicher als erstes [TrueCrypt](#) ein – was für Android noch nicht so lange verfügbar ist; allenfalls konnte man *TrueCrypt*-Container auf dem Androiden ablegen, um so einen "portablen, sicheren Speicher" bei der Hand zu haben (wie man das für besonders große Container umsetzt, beschreibt [dieser Artikel](#)).

Vom Verhalten am Nächsten kam dem bislang der [LUKS Manager](#) (rechtes Bild), der AES-verschlüsselte Container verwaltet. Diese können wie andere Dateisysteme eingebunden werden, und stehen dann ebenso anderen Apps zur Verfügung. *LUKS* kümmert sich dabei sowohl um die Erstellung/Entfernung als auch das Einbinden/Aushängen der entsprechenden Container.

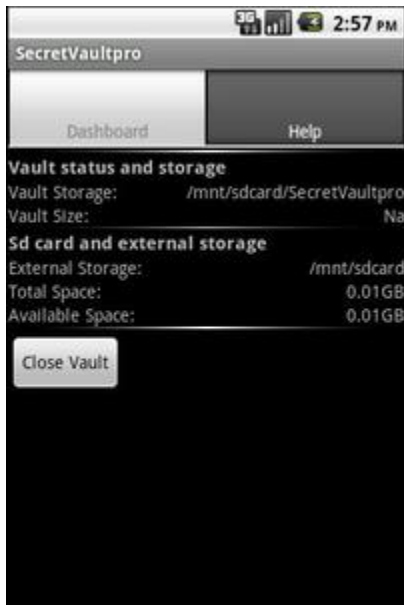
Die App ist zwar gratis – doch sind die Hürden für den "einfachen Anwender" ein wenig hoch. So werden nicht nur [root](#) inkl. der App *BusyBox* vorausgesetzt – auch ein passender Kernel, der DM_CRYPT und "Loopback" unterstützt, muss vorhanden sein (beides Dinge, die auf die meisten Standard-Kernels/Standard-ROMs nicht zutreffen). Womit *LUKS* für die meisten Android-User leider als Option ausfällt...

Alternativ gibt es seit ein paar Monaten [Eds](#). Und diese App arbeitet mit TrueCrypt-kompatiblen Containern. Auch ist von den bei *LUKS* genannten Hürden hier an keiner Stelle zu lesen. Sogar Dropbox-Support ist direkt aus der App



(Vollversion) heraus möglich, sodass man seine verschlüsselten Container in der Cloud ablegen kann – das mindert die Bedenken gegen "Schnüffler im Netz", da diese ja in den Container nicht hineinschauen können.

Eds gibt es in einer Gratisversion (ohne Dropbox-Support und mit minimalen Rechte-Anforderungen, u.a. kein Netzwerkzugriff), sowie in einer Vollversion für derzeit ca. fünf Euro. Beide Varianten sind sehr gut bewertet.



Womit wir zu [SecretVault](#) (linkes Bild) als nächster Alternative kommen. Anders als bei *LUKS* handelt es sich hier nicht um Open Source – wodurch die auch in der Kaufversion geforderte Berechtigung zum Internet-Zugriff manch einem ein wenig Bauchschmerzen bereiten dürfte. Selbst in der Pro-Version lässt sich nur ein einziger Container verwenden, der mit der App selbst im Basisverzeichnis der SD-Karte erstellt wird. Wurde dieser Container eingebunden, steht er transparent als Verzeichnis im System zur Verfügung – bis die App selbst beendet oder der Container aus der App heraus geschlossen wurde.

Detailliertere Informationen, eine Reihe weiterer Screenshots sowie eine kurze Beschreibung des Umgangs mit *SecretVault* finden sich im [Blog der App](#).

Damit wäre der Vorrat an "transparenten Container-Apps" allerdings leider bereits erschöpft. Es verbliebe aber noch die Möglichkeit, Dateien und Verzeichnisse separat zu verschlüsseln. Der Favorit hierfür dürfte [Droid Crypt](#) (Bild rechts) heißen, und verschlüsselt wahlweise einzelne Dateien oder ganze Ordner (auf Wunsch gleich rekursiv) mittels [AES](#). Auch diese App vermag es, sich als Mittler zwischen den Apps und den verschlüsselten Daten zu betätigen – und so eine gewisse Transparenz zu ermöglichen. Der Anwender muss also nicht erst die benötigten Daten von Hand entschlüsseln, um sie mit einer App zu bearbeiten (und nicht vergessen, sie hinterher wieder zu verschlüsseln). Wer mag, kann zusätzlich eine Datenkompression vornehmen lassen: Auf diese Weise spart man zusätzlich Speicherplatz.

Interessant ist auch die Wahl des Passwortes. Oder auch nicht – denn ein solches ist nicht zwingend erforderlich. Stattdessen lässt sich auch ein "Schwenkmuster" verwenden.

Die Testversion von *Droid Crypt* verlangt noch den Zugriff auf's Internet für Werbeeinblendungen; nach 60 Tagen Testzeitraum ist sie darüber hinaus auf die Ver- und Entschlüsselung von einzelnen Dateien beschränkt. Beides gilt natürlich



nicht für die Vollversion, die man bereits für ca. zwei Euro erwerben kann. Eine kurze Liste weiterer möglicher Kandidaten für die Ver-/Entschlüsselung von Dateien und Verzeichnissen findet sich [im Forum](#).

Ab Android 3.0 (aka *Honeycomb*) auf Tablets bzw. 4.0 (aka *Ice Cream Sandwich*) auch für Smartphones (für Besitzer des Samsung Galaxy S2 bereits ab Android 2.3 *Gingerbread*) werden all diese Umwege für die meisten Anwender unnötig werden: Hier ist eine Verschlüsselung des gesamten Gerätes von Haus aus mit dabei. Zum Einsatz kommt dafür [dm-crypt](#) mit 128-Bit [AES](#)-Verschlüsselung. Der Anwender muss dann bei jedem Bootvorgang das zugehörige Kennwort zur Entschlüsselung des Dateisystems eingeben – ab diesem Zeitpunkt stehen die Daten dann allerdings unverschlüsselt zur Verfügung.

Apps mit Passwort absichern

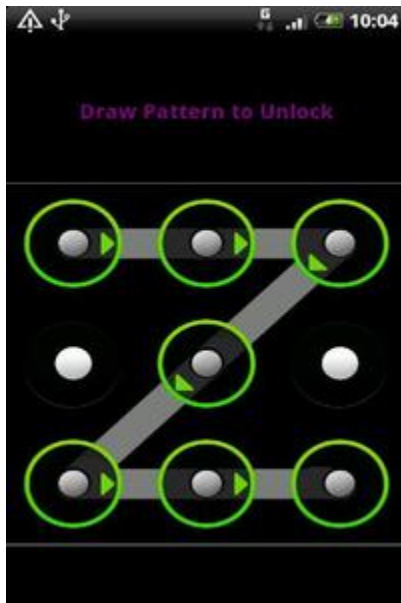
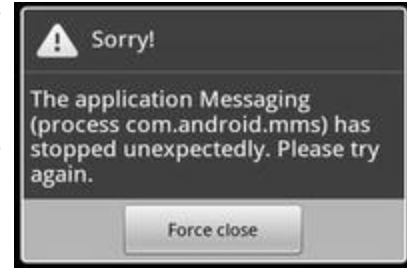
Viele Gründe kann es geben, warum man so etwas tun möchte: Etwa wenn man sein Handy "mal eben" einem Kollegen "nur für einen kurzen Anruf" übergibt. Oder es einem Freund "für ein paar Tage" borgt, während dessen Telefon sich "in Reparatur" befindet. Darüber hinaus kann es sich als sinnvoll erweisen, seinem (vorpubertären?) Kinde die Nutzung des Smartphones zwar prinzipiell nahezubringen – aber dennoch nicht zuletzt aus Sicherheitsgründen etwas einzuschränken.



Im letztgenannten Fall bietet sich insbesondere [Child App Protector](#) (Bild links) mit einigen nützlichen Einstellungen an. Eine davon zeigt die Abbildung bereits: Die Nutzung bestimmter Apps (etwa Spiele oder Chat-Apps) lässt sich zeitlich limitieren – damit sich der "Zögling" nicht ausschließlich damit beschäftigt. Aber vielleicht möchte man bestimmte Dinge ja komplett abschalten – oder sicherstellen, dass gewisse Einstellungen nicht verändert werden? Auch das stellt die App nicht vor Probleme: Ausgewählte Apps lassen sich gegen einen Start schützen, solange *Child App Protector* aktiv ist (hier sollte man einen etwaigen Task-Killer nicht vergessen). Schmäckerl: Man kann je App eine "alternative App" festlegen. Wird nun versucht, die gesperrte App zu starten – startet stattdessen die Alternative (auch "erzieherische Maßnahme" genannt). Zu guter Letzt lässt sich auch ein Backup

des Homescreens anfertigen, bzw. bei "ungewollten Änderungen" an selbigem wieder herstellen.

Damit wären eigentlich alle Fälle abgedeckt. Aber dennoch gibt es einige Alternativen, die genannt werden möchten. Weniger in Sachen Kinderschutz: Das Thema ist ja vielmehr der Schutz von Apps. Und da hat sich [App Schutzeinrichtung](#) (oder besser deren Entwickler) etwas besonders nettes einfallen lassen: Versucht jemand, eine geschützte App zu starten – so sieht er so etwas, wie da rechts zu sehen ist. Es wird ein Systemfehler vorgegaukelt – so, als sei die betreffende App halt einfach kaputt. Dem unberechtigten Nutzer wird also weisgemacht: Du brauchst es gar nicht weiter probieren, das Ding ist im Eimer. Und somit von der eigentlichen "Protection" abgelenkt...



Als weitere Alternative wäre noch [Application Protection](#) (linkes Bild) zu nennen – von fast achttausend Anwendern mit durchschnittlich 4,4 Sternen bedacht, kann diese App ja nicht schlecht sein. Apps lassen sich hier wahlweise mit einem Sperrmuster oder einem Code schützen. Dabei wird davon ausgegangen: Wurde der korrekte Code einmal eingegeben, handelt es sich wohl auch um den "rechtmäßigen Besitzer" – bis zur nächsten "Scharfschaltung" (die bei Neustart des Gerätes automatisch eintritt) bleibt man daher von weiteren Anfragen verschont. Dies lässt sich konfigurieren, und kann so pro App oder global aktiviert werden.

Übrigens: Wird bei dieser App zehn Mal ein falscher Code eingegeben, verschickt sie automatisch eine Mail an die in der App konfigurierte Adresse. Der Beschreibung nach lässt sich darauf schließen, dass diese Mail auch den korrekten Sperr-

Code enthält – gut für den Fall, dass man ihn selbst einmal vergessen hat. Für diesen Fall lässt sich allerdings auch ein Merksatz hinterlegen...

Wem nun diese kurze Zusammenstellung noch nicht genügt, der findet in [dieser Übersicht](#) noch weitere Kandidaten.

Zugriffsrechte

Permissions werden oftmals zu wenig beachtet oder falsch interpretiert. Nur wenige scheinen ihre Bedeutung überhaupt *richtig* zu verstehen und korrekt mit ihnen umzugehen – das betrifft nicht nur die Anwenderseite, sondern ebenso die Entwickler. Wofür sind diese "Permissions" eigentlich gut, was nützen sie? Um das zu verstehen, müssen wir einen kurzen Blick hinter die Kulissen werfen.

Android baut bekanntermaßen auf Linux auf. Hier verwaltet das System u. a. Benutzer und Benutzergruppen, um die "Eigentumsverhältnisse" zu klären: Auf die Daten eines anderen Benutzers kann somit nicht ohne weiteres zugegriffen werden. Während unter Linux ein Benutzer vereinfacht betrachtet einer am Rechner angemeldeten Person entspricht, teilt Android jeder App eine eigene

Benutzer-ID zu. Somit kommt eine App in der Regel nicht ohne weiteres an die Daten der anderen Apps heran. Ist sie mit keinerlei expliziten Permissions ausgestattet, kann sie nur "völlig ungefährliche" Aktionen ausführen: Etwas auf dem Bildschirm anzeigen, eigene Daten speichern, und ähnliche Dinge.

Für den Zugriff auf Spezielleres muss der Entwickler seine App entsprechende Permissions anfordern lassen – und zwar bereits bei der Installation (im Nachhinein ist dies auf normalen Wegen nicht mehr möglich). Am bekanntesten ist hier sicher der Zugriff auf das Internet. Aber auch der Zugriff auf persönliche Daten wie Kontakte und Termine, das Benutzen von Hardware-Elementen wie Kamera und Vibrator, oder gar das Anpassen von Systemeinstellungen bedarf expliziter Genehmigung. Eine Auswahl verfügbarer Permissions und ihrer Bedeutung findet sich im Anhang [Google Permissions](#).

Wo liegen nun die größten Fehler im Umgang mit diesen Permissions?

Bei Anwendern am ehesten darin, dass sie diese häufig ohne genaueres Hinschauen abnicken, weil sie die App ja haben wollen. Dies liegt nicht zuletzt in einem Schwachpunkt des Android-Systems begründet: "Alles oder nichts" – entweder alle geforderten Permissions werden akzeptiert, oder die fragliche App kann halt nicht installiert werden. Wünschenswerter wäre es hier, einzelne Permissions gezielt ausklammern zu können. Das würde zum Einen die Anwender motivieren, sich überhaupt einmal mit ihnen zu beschäftigen – und zum Anderen so manch einer Malware die Grundlage entziehen. Möglich ist dies derzeit nur, sofern man über [root](#)-Rechte verfügt, oder die entsprechenden Apps modifiziert.

Bei Entwicklern ist der Fehler häufig, dass sie zu viele Permissions anfordern. Entweder aus Unwissen, aus Bequemlichkeit ("so kann ich nichts vergessen") – oder mit dem Hintergedanken: Wenn das später mal gebraucht wird, und ich es erst dann hinzufüge, bekomme ich nur wieder böse Kommentare...

Was sollte man also im Umgang mit Permissions besser beachten?

Der Anwender sollte sich vor der Installation einer App genauer fragen: Machen die geforderten Permissions für die gebotene Funktionalität Sinn? Sicher benötigt eine SMS-App Zugriff auf die Kurznachrichten, und muss selbige auch senden können (auch wenn dies natürlich Kosten verursachen kann). Sie muss aber weder Telefonanrufe tätigen, noch den Netzwerkstatus ändern können. Eine Wallpaper-App wiederum hat an den Kurznachrichten gar nichts zu suchen. Die Alarmglocken läuten in den lautesten Tönen, wird etwa Permission zur Installation weiterer Packages gefordert (was allerdings i. d. R. ins Leere laufen dürfte, da an dieser Stelle weitere Sicherheitsmaßnahmen des Systems greifen).

Sicher gibt es hier Grenzfälle: Besagte SMS-App könnte das Internet zum Versenden kostenloser/kostengünstiger SMS über entsprechende Web-Services nutzen. Sie benötigt natürlich auch Zugriff auf die Kontakte (wohin schickt man wohl SMS?). Und spätestens hier taucht die Frage auf: Könnte sie dann eine Kopie der kompletten Kontaktliste an eine dubiose Website verschicken? Die Antwort: Ja, sie könnte – sie hat schließlich sowohl vollständigen Zugriff auf die Kontakte (lesend genügt ja), als auch auf das Internet. Da kann man wohl nichts machen? Oh doch: Auf den Entwickler zugehen, damit er sich den folgenden Abschnitt zu Herzen nimmt:

Entwickler sollten natürlich ihre Apps nur die Permissions anfordern lassen, die wirklich benötigt werden. Das lässt auch ihre Apps vertrauenswürdiger aussehen. Bei "gefährlichen Kombinationen" wie "Kontaktdaten lesen & Internet" sollte,

sofern möglich, die Auslagerung in ein AddOn in Erwägung gezogen werden. Positives Beispiel: [Locus Maps](#). Sicher macht hier ein Zugriff auf die Kontaktdaten Sinn: Um deren Adresse auf der Karte anzeigen zu können. Für den Download von Kartenmaterial wird allerdings der Zugriff auf das Internet benötigt. Der Entwickler entschied sich also, den Kontaktdaten-Zugriff in ein eigenes AddOn ([Locus - addon Contacts](#)) auszulagern. Somit sind die Permissions der Kern-App wieder "unanständig" – und dem Anwender steht es frei, die App mit oder ohne diese "Bequemlichkeit" des Zugriffs auf die Kontaktadressen zu nutzen. Vorbildlich, Menion! Mögen sich viele Entwickler daran ein Beispiel nehmen!

Permissions überprüfen



Leider kommt es oftmals vor, dass man die Liste der geforderten Permissions vor der Installation eben *nicht* (oder nicht genau) gelesen hat – obwohl man es sich fest vorgenommen hatte. Oder, dass die Einsicht erst beim Lesen dieses Buches kam – und natürlich bereits allerhand "unbedacht installiertes" auf dem Androiden "herumliegt". In beiden Fällen ist eine nachträgliche Überprüfung geboten.

Was nicht immer einfach ist: Bei den vielen unterschiedlichen Permissions ist nicht immer klar, was sie eigentlich bedeuten. Wer hier einmal alles ein wenig genauer abklopfen will, findet unter Umständen in [RL Permissions](#) (linkes Bild) das Werkzeug seiner Wahl. Mit dieser App lassen sich zum Einen alle Apps auflisten, die eine gewählte Permission verlangen – oder umgekehrt alle Permissions, die eine bestimmte App verlangt. Dabei wird zu den Permissions auch jeweils eine Erklärung geliefert, was diese bedeuten. Ob von ihnen eine potentielle Gefahr ausgeht, besagt auch die kleine jeweils neben ihnen angezeigte Ampel. Auch hier heißt "rot" nicht gleich, dass eine App "böse" ist – sondern lediglich, dass sie das potentiell sein *könnte*.

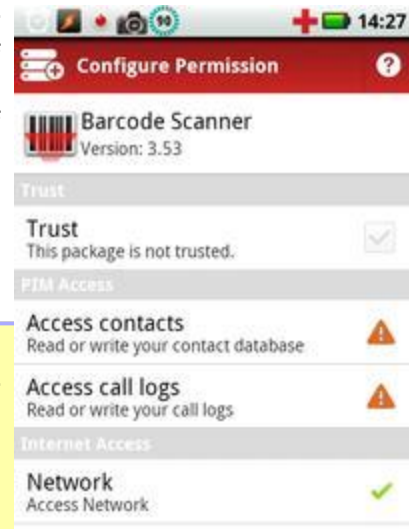
Es gibt natürlich noch weitere Apps, die ähnliches tun – einige davon sind in [diesem Artikel](#) näher vorgestellt.

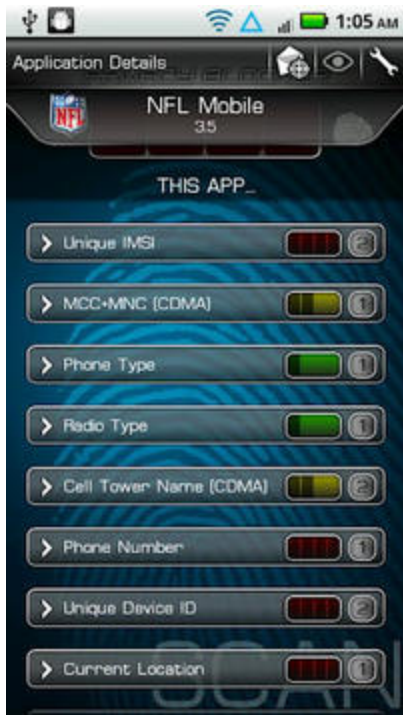
Permissions überwachen

Eine echte Überwachung einschließlich der Möglichkeit, einer App bei "unbefugtem Zugriff" auf die Finger zu hauen – das ist ohne root leider nicht möglich. Denn dazu bedarf es ja eines Zugriffs auf die Aktivitäten der jeweiligen App – eine Sache, die das Berechtigungssystem so nicht erlaubt. Schade daher, dass die entsprechende Möglichkeit nicht von Haus aus ins System integriert ist; eine solche Schutzfunktion ist aus meiner Sicht unerlässlich.

Dass ich mit dieser Meinung nicht allein dastehe zeigt sich auch darin, dass einige Entwickler sich der Sache bereits angenommen haben. So erlaubt etwa das Custom ROM von Cyanogen, Apps einzelne Permissions zu entziehen. Und es gibt die App LBE Privacy Guard (rechtes Bild), welche dies dynamisch erlaubt. Die App teilt auch in der Notification Area mit, wenn eine neue App installiert wurde – und bietet die Konfiguration einiger Permissions an: So lässt sich beispielsweise der von ihr gewünschte Zugriff auf Netzwerk und Systemlogs unterbinden, während beim Zugriff auf Kontakte eine Rückfrage erfolgen soll ("Die App XYZ möchte... Darf sie das?").

Die Voreinstellungen lauten hier: Nachfragen. Dabei erfolgt nach 15 Sekunden ein "Timeout", falls man die Abfrage verpasst (etwa weil die App den Zugriff im Hintergrund, bei ausgeschaltetem Display, tätigte). In diesem Fall wird der entsprechende Zugriff unterbunden. Und das scheinbar recht intelligent, denn mir ist dabei noch keine einzige App abgestürzt! Ich vermute daher, dass LBE hier Daten faked: Das Adressbuch ist dann beispielsweise leer, das Internet gerade nicht verfügbar, und die IMEI ein Fantasiewert. Gut gemacht – nur braucht es root...





Hat man ohne *root* denn wirklich nicht die geringste Chance? Ganz so ist es zum Glück nicht, denn es gibt beispielsweise noch [Privacy Blocker](#) (linkes Bild). Ohne *root* kann auch diese App natürlich nicht "in den laufenden Betrieb" eingreifen – daher ist das Vorgehen hier ein vollständig anderes: Der Benutzer wählt zunächst die gewünschte App (die mit den unerwünschten Permissions) aus. *Privacy Blocker* greift sich daraufhin die [APK-Datei](#) dieser App, nimmt sie auseinander, und präsentiert dem Anwender die Liste der von ihr geforderten Permissions (siehe Screenshot). Hier lassen sich nun "unerwünschte" Permissions auswählen – und schließlich wird eine neue APK-Datei erzeugt, die eben diese nicht mehr verlangt. Selbige kann man nun installieren.

Klingt soweit eigentlich ganz gut, nur hat die Sache zwei "kleine" Haken: Zum Einen eignet sich die so erzeugte APK-Datei nicht für das Update einer bereits installierten Version (die Signatur hat sich geändert – Android hält es also für eine andere App, die dummerweise nur gleich heißt). Zum Anderen wird die App nun mit großer Wahrscheinlichkeit abstürzen, wenn sie auf eine Funktionalität zugreift, für die sie nun keine Permission mehr hat: Da der Entwickler (mit Recht) davon ausgehen konnte, dass ihm der Zugriff erlaubt wird, hat er diesen "Zugriffsfehler" wahrscheinlich nicht abgefangen. Bei einem "bösen" Zugriff ist dies mit Sicherheit die bessere Alternative. War der Zugriff aber doch funktional erwünscht, weiß man nun zumindest Bescheid, wozu diese Permission benötigt wird – und installiert entweder die Originalversion, oder baut sich mit *Privacy Blocker* eine weniger restriktive Fassung.

Die App ist im Playstore für ca. anderthalb Euro verfügbar. Eine gratis Testversion gibt es ebenfalls; diese beschränkt sich jedoch auf die Anzeige, geänderte APK-Dateien lassen sich damit nicht erstellen.

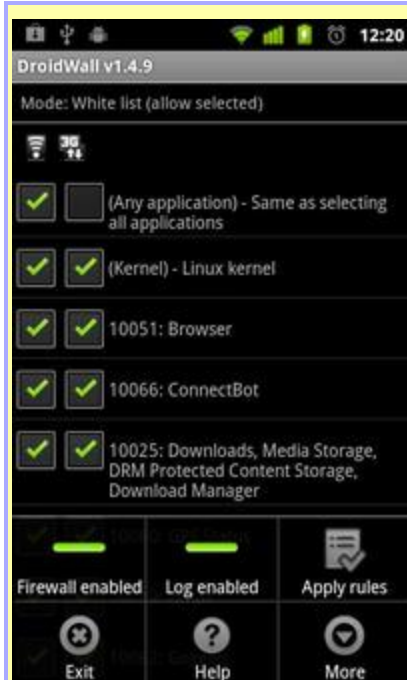
Weitere Apps zu diesem Thema finden sich wiederum bei AndroidPIT in einer [Übersicht](#) aufgeführt.

Firewalls

Auch sogenannte "Personal Firewalls" fallen in die Rubrik "Zugriffsrechte" – dienen sie doch dazu, den Zugriff auf bestimmte Dinge zu beschränken. Wieder einmal gilt die übliche Einschränkung: "Ohne [root](#) sich nichts tut", denn so etwas greift tiefer ins System ein.

Um den Bereich "Bluetooth" kümmert sich die rechts abgebildete [Bluetooth Firewall](#) - und benötigt dafür ausnahmsweise keinen *root*. Häh? Blauzahn-Attacken? Blauwal, auf See, klar. Aber was kann denn schon... Ho-ho-ho. Da kann. Jemand an den Einstellungen drehen, oder die Kontaktliste auslesen, oder... Ja, kann böse sein. Und was tut so eine Firewall? Sie sperrt halt alles unerwünschte aus. Oder ein. Je nachdem - eine Verbindung kann ja immer von zwei Seiten her aufgebaut werden.

Und so schlägt die *Bluetooth Firewall* beispielsweise Alarm, wenn eine App mit Bluetooth-Fähigkeiten installiert bzw. aktualisiert wurde, oder Bluetooth-Aktivitäten starten will. Ebenso aber auch, wenn jemand von außen per Bluetooth zugreifen möchte. Außerdem ermöglicht sie einen Bluetooth-Scan (welche Bluetooth-Geräte sind in Reichweite?), protokolliert alle Bluetooth-Aktivitäten, und gibt dem Anwender die Möglichkeit, ausgewählte Bluetooth-Geräte als "vertrauenswürdig" einzustufen.



Für die Internet-Verbindungen hingegen ist [DroidWall](#) (linkes Bild) zuständig. Genau genommen handelt es sich hier "nur um eine Eingabemaske" zur Erfassung von Regeln - die eigentliche Arbeit macht *iptables*, welches hoffentlich bereits auf dem Androiden installiert ist - denn dies ist natürlich Voraussetzung für die Funktionalität. Aber leider nicht auf jedem Androiden gegeben - sollte *iptables* nicht vorhanden sein, hilft daher nur ein [Custom ROM](#).

Ein schönes Feature von *DroidWall* ist auch, dass sich der Zugriff getrennt nach Netzwerk-Interface steuern lässt: So kann man beispielsweise einer datenhungrigen App die Nutzung des "mobilen Internets" verbieten, und sie auf WLAN-Verbindungen beschränken - während man (vermeintlichen) "Schnüfflern" den Netzzugang komplett untersagt.

Übrigens zeigt der Screenshot noch etwas anderes recht eindrücklich: Die "Zahlen" vor den Namen der aufgeführten Apps bezeichnen die "User-ID" derselben. Womit man sieht, was ich eingangs bereits beschrieb: Unter Android bekommt jede App eine eigene User-ID. Und über diese kontrolliert auch *DroidWall* den Netzzugriff.

Location-Cache

War da nicht was? Legte da nicht wer Bewegungsprofile seiner Nutzer an? Potentiell möglich wäre so etwas auch unter Android: Möchte man hier beispielsweise die "netzwerk-basierte Standortermittlung" aktivieren, erfolgt prompt der Hinweis, dass man dann auch die damit verbundenen Informationen "freiwillig" an Google weitergibt. Und spätestens seit Android 2.2 heißt es auch hier: "Alles oder nichts". Also keine "netzwerk-basierte" (und damit Akku-schonende) Lokalisierung, ohne "Big Brother" zu informieren, wo man ist.

Der Zusammenhang dürfte wahrscheinlich sein, dass Google gern über "neue" oder "umgezogene" Zellen (Mobilfunk und WLAN) informiert sein möchte, um diese bei weiteren Ortungs-Anfragen nutzen zu können – schließlich erfolgt die Zuordnung des Standortes ja über eine von Google bereitgestellte Datenbank. Einen faden Beigeschmack hinterlässt dies trotzdem – nicht zuletzt, weil der Grund dafür gar nicht mitgeteilt wird. Man darf also raten.

Hat man diesen Ortungsdienst nun aktiviert, werden alle Ortungsdaten "gepuffert", d. h. in einem Zwischenspeicher (auch "Cache" genannt) abgelegt. Ab und an kommt dann ein kleines Google-Dienerlein vorbei, packt den Inhalt zusammen, und verschifft die Daten in Googles Rechenzentrum. Ist also nichts im Cache, wird auch nichts verschickt – einen Sachverhalt, den **Location Cache** (rechtes Bild) zu unseren Gunsten ausnutzen kann.



Die rot beschrifteten Buttons lassen es unschwer erkennen: Die App kann den Cache leeren, und sogar blockieren. In letzterem Fall wird er sozusagen "schreibgeschützt", sodass keine neuen Daten darin abgelegt werden können. Das ist effektiv: Die netzbasierte Ortung funktioniert weiterhin, die Datenkrake aber bleibt ungefüttert. Dafür dauert die erste Anzeige auf der Karte dann ggf. ein wenig länger, da die Daten ja nicht dem Cache entnommen werden können.



Eine weitere Möglichkeit hat die App mit dem links abgebildeten [Android Location Cache Viewer](#) gemeinsam: Bei gefülltem Cache können uns beide Apps mitteilen, wo wir in letzter Zeit gewesen sind. Dabei stellt man erfreulicherweise fest, dass der Füllstand des Caches recht begrenzt ist – und zwar auf 50 Mobilfunk-Zellen sowie 200 WLAN-Netze.

Beide Apps können die im Cache gespeicherten Locations übrigens auf der Karte anzeigen, und erlauben auch den Export der Daten im GPX-Format. Damit wir zumindest unsere eigenen Bewegungs-Profile erstellen können. Ebenfalls für beide Apps gilt wieder die übliche Einschränkung beim Eingriff in System-Internia: [root](#)-Zugang zum Gerät ist Grundvoraussetzung...

Malware und Viren

Zu diesem Thema gibt es ja fast jede Woche (zumindest gefühlt) einen Artikel in den News. In Bild-Deutsch gesprochen: "140% aller Android-Apps sind mit Malware und Viren verseucht!". Oder so ähnlich. Meist kommen diese Warnungen von wohlbekannten Firmen aus dem Sicherheitssektor, mit Namen wie Kaspersky oder McAfee. Moment, woher kommen uns diese Namen bekannt vor? Richtig: Diese Firmen bestreiten ihren Umsatz mit dem Verkauf von Anti-Virus und Anti-Malware Produkten. Da ist sicher die Frage erlaubt: Wollen die nur den Umsatz ankurbeln – oder ist an den Meldungen etwas dran?

Die Wahrheit liegt, wie so oft, in der Mitte: Natürlich will man den Umsatz ankurbeln. Trotzdem handelt es sich um Fakten – es wird also nicht gelogen. Nur ein wesentlicher Teil der Wahrheit im Kleingedruckten versteckt...

Was jedoch bei genauerem Lesen mit ein wenig Hintergrundwissen (welches mit den vorigen Kapiteln aufgebaut sein sollte) auffallen sollte. Denn was macht diese böse Malware? Als Beispiel nennt McAfee da eine Kalender-App, die heimlich Premium-SMS verschickt. Was haben wir jedoch bereits im Kapitel [Zugriffsrechte](#) gelernt: Vor der Installation prüfen, ob Berechtigungen Sinn machen. Wo macht bitte die Permission zum SMS-Verschicken bei einer Kalender-App Sinn? Eigentlich hätte sie bereits an der Stelle auffallen müssen. Bei vielen der anderen genannten Beispiele verhält es sich ähnlich; es kam mir bislang kein Beispiel unter, welches sich nicht in dieses Muster eingepasst hätte. Und leider gehen die Sicherheits-Apps der namhaften Hersteller, was das Anfordern von Permissions betrifft, nicht gerade mit gutem Beispiel voran – schon mehrfach musste ich dabei denken: Liebe Leutz, nach Euren eigenen Worten dürfte ich Eure App gar nicht installieren... Und wie um noch eins obendrauf zu setzen, schreibt TelTarif: [Smartphone-Viren sind oft nur heiße Luft](#). Man kann sagen, dass die Antiviren-Software-Hersteller ein großes Interesse haben, die Gefahr groß aussehen zu lassen, manchmal auch größer als sie wirklich ist (ebenda).

GMV

Womit ich bei einem meiner Lieblings-Themen angekommen bin: Seit ich [GMV](#) zum ersten Mal in einem Post bei AndroidPIT und dann auch im AndroidPITiden-Buch genannt habe, bekam ich des Öfteren Anfragen: Wo könne man diese Android-App denn herunterladen? Auch Google-Suchanfragen nach "GMV App Android" fanden schon den Weg auf meinen Server. Dabei braucht man doch nur dem Link zu folgen...

Also noch einmal im Klartext: Bei GMV handelt es sich *nicht* um eine Android-App. GMV sollte im *biologischen Speicher* (auch als "Brain 2.0" bekannt) *vorinstalliert* sein. Die Abkürzung steht nämlich für "**G**esunder **M**enschen**v**erstand". Und den hat doch hoffentlich jeder, zumindest in gewissem Umfang. Natürlich kann man sich alle möglichen Anti-Viren und Anti-Malware Apps auf seinem Androiden installieren – doch das Einzige, was damit garantiert ist, sind belegter Speicherplatz und Ressourcen-Verbrauch. Keiner der Hersteller wird garantieren, dass damit die Gefahr zu 100% gebannt ist, dass jeder Schädling erkannt und erfolgreich "abgeschossen" wird. Im Gegenteil las ich vor ein paar Wochen im Forum von einem Selbstversuch: Jemand installierte sich eine dieser Anti-Virus-Anti-Malware Apps, und versuchte anschließend eine Warnung zu provozieren – doch die gab keinen Laut, obwohl er sich richtig böse klingende Sachen herausgesucht hatte (leider finde ich den Post nicht mehr – ersatzweise lohnt aber auch ein Blick in [diesen Blog-Eintrag](#)).

Auf was sollte man also bei der Installation einer App achten? Ich zitiere hier einmal ganz dreist aus meinem AndroidPITiden-Buch:

- Ist die Quelle vertrauenswürdig?
 - Positiv-Beispiele: AndroidPIT-Market, AppCenter, Google Playstore, Website des bekannten (!) Entwicklers
 - Negativ-Beispiele: Bei Rapidshare "gefunden", in einer Tauschbörse aufgetrieben, per eDonkey aus unbekannter Quelle gezogen...
- Sehen die Permissions vernünftig aus?
 - Positiv-Beispiele: Ein Webbrowser muss ins Web, eine SMS-App kann natürlich SMS lesen/schreiben/schicken und braucht ggf. auch (lesend) Zugriff aufs Adressbuch
 - Negativ-Beispiele: Eine Wallpaper-App braucht in der Regel keine Telefonnummern anrufen, ein Ballerspiel muss keine SMS senden.
 - Besondere Vorsicht: Apps, die auf persönliche Daten (Kontakte, Kalender, Nachrichten) zugreifen und gleichzeitig ins Internet wollen. Leider lässt sich bei letzterem (Internet) die Frage der Notwendigkeit nicht so einfach beantworten – es könnte auch einfach nur für Werbung-Laden gebraucht werden...
- Was sagen andere Nutzer zur App/zum Entwickler (Bewertungen, Forum)?
 - Auch hier wieder GMV aktivieren. Kommentare wie "Geil!", "Super", etc. sagen nicht wirklich etwas aus (da hat eher jemand bei deaktiviertem GMV einen Kommentar hinterlassen)
 - Gleiches gilt für manchen negativen Kommentar: Nicht gerade selten passiert es, dass jemand einfach zu blöd war. Oder die Anforderungen der App gar nicht verstanden hat.

- Nicht alle Bewertungen beziehen sich wirklich auf die App. Die kann schließlich nix dafür, wenn der Playstore mal wieder klemmt, und daher der Download nicht funktioniert. Oder die HD-Video-App, die mindestens WVGA benötigt, mit dem Motorola Flipout (mini-Display) im Playstore nicht gefunden wird...
- Ganz neue App? Noch keine Bewertungen? Im Zweifelsfall im Forum nachfragen, ob schon jemand die App kennt und etwas dazu sagen kann.

Wer diese Punkte beachtet, hat eigentlich bereits mehr als die "halbe Miete" eingefahren. Dann noch Vorsicht mit diversen Werbebannern, die einen mit Abos "beglücken" wollen - und ein 90%iger Schutz sollte gegeben sein. Natürlich können andere Apps aus der "Sicherheits-Abteilung" eine gute Ergänzung zu GMV bieten. Insbesondere bei [Verlust des Gerätes](#) - denn dagegen macht auch GMV nicht immun...

Rundum-Sorglos-Pakete

Die sollen einen wohl in Sicherheit wiegen - versprechen sie doch, sich um alles zu kümmern: Viren, Malware und Diebe sollen gleichermaßen eins auf die Mütze bekommen. Interessant ist hier: Nach einem Blick auf die verlangten Permissions müssten viele dieser Apps eigentlich gleich selbst draußen bleiben. Wozu bitte muss [Lookout Security Antivirus](#) (rechtes Bild) denn bitte Synchronisierungseinstellungen lesen/schreiben, Kontaktdaten lesen/schreiben, Kalenderdaten schreiben, Benutzerwörterbuch lesen/schreiben... Das Ding soll sich doch um die Abwehr der "bösen Jungs" kümmern, und nicht selbst an diesen persönlichen Dingen herumbasteln! Das kommt mir vor, als wolle man den Teufel mit dem Beelzebub austreiben.

Wer jetzt glaubt, die Antwort darauf in der App-Beschreibung zu finden: Fehlanzeige. Die Mühe, die ganze Website hinter dem angegebenen Link zu durchsuchen, habe ich mir nicht gemacht - zumindest in den dortigen FAQs findet sich auch keine Antwort. Ein Durchschnitt von 4,6 Sternen bei über 200.000 Bewertungen stimmt hingegen beruhigend: Offensichtlich wird mit den genannten Permissions wirklich nichts "böses" angestellt. Allerdings sollten sich die Entwickler solcher Software auf die Fahnen schreiben, hier mit gutem Beispiel voran zu gehen, anstatt die Anwender zu verunsichern.

Davon abgesehen klingt *Lookout* eigentlich recht vielversprechend: Jede heruntergeladene App wird hier auf "Schädlingsbefall" geprüft. Zusätzlich kann man "Sicherheits-Komplettscans" seines Gerätes einrichten, die automatisch zum angegebenen Zeitpunkt (etwa wöchentlich) starten.

Das Gerät ist abhanden gekommen? Verlegt oder gestohlen? Per Google Maps lässt es sich lokalisieren (auch bei abgeschaltetem GPS; es wird dafür jedoch ein Account bei myLookout.com benötigt). Auch ein lauter Alarm vom Gerät selbst



kann ausgelöst werden – sogar wenn es gerade auf "lautlos" steht (da eignet sich bestimmt eine MP3, wo jemand laut "DIEBE!" schreit...). Als Dreingabe gibt es noch Backup und Restore wichtiger Daten. Das Schöne an der Sache: All dies ist gratis.



Auch die Großen im Bereich Antivirus & Co. sind hier natürlich vertreten – wie etwa Kaspersky mit seiner [Kaspersky Mobile Security](#) (linkes Bild) oder Symantec mit der [Norton Mobile Security](#). Beide dürfen sich die gleichen Vorwürfe bezüglich der verlangten (und nicht begründeten) Permissions machen lassen. So möchte die Norton-Suite u. a. gern irgendwo anrufen, lesend und schreibend auf die Kontaktdaten zugreifen, und so weiter. Bei Kaspersky sieht es nicht besser aus: Auf Kontaktdaten und Kalender möchte die App lesend und schreibend zugreifen (sicher für den Vermerk, wer das Gerät wann gestohlen hat). Anrufen möchte auch diese App irgendwo, sowie an den Synchronisierungseinstellungen und APN-Einstellungen basteln. Bei einer solchen Kombination läuten eigentlich bei mir sämtliche Alarmglocken und schreien: Vorsicht, Malware!

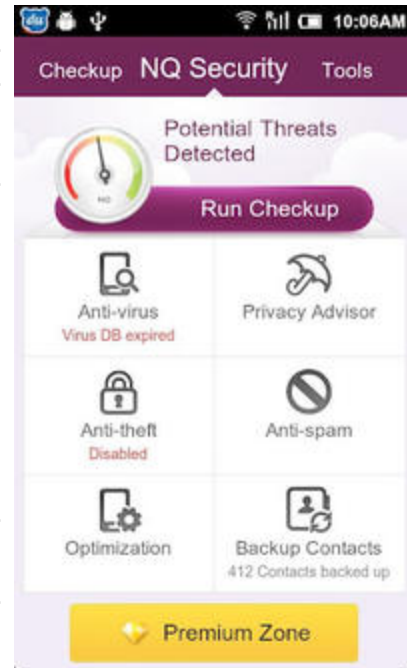
Was aber bringt Eugen Kaspersky uns nun mit seiner App – bzw. was verspricht er uns als Gegenleistung für die gut sieben Euro Kosten? Echtzeit sowie "On-Demand" Scans nach Viren und Malware, Privacy für Kontakte (bestimmte Kontakte markieren, sodass keine Spuren der Kommunikation mit ihnen zu sehen sind – aha, das erklärt zumindest die Zugriffe auf die Kontaktliste), Anruf- und SMS-Filter, Diebstahl-Schutz (Lock, Wipe, Anzeige einer Nachricht wie z. B. "Ich wurde verloren/gestohlen, bitte xxx anrufen", Anzeige der Location in Google Maps, sowie Email-Alert mit neuer Rufnummer bei SIM-Wechsel – sofern der Dieb nicht schlau genug war, vorher einen Wipe zu machen).

Genauer erkundigen sollte man sich auch noch, ob zu den gut sieben Euro für die App selbst noch jährliche Kosten von 25 Euro hinzukommen, wie verschiedentlich im Forum berichtet wurde.

NetQin Security & Anti-virus (rechtes Bild) gehört ebenfalls hierher – bietet es doch gratis nicht nur Schutz vor Viren und Malware, sondern, wie der Screenshot rechts zeigt, auch einen Diebstahlschutz und sogar einen Traffic-Manager. Hinzu kommt auch noch ein System-Monitor mit Anzeige der Speicherauslastung und Taskkiller sowie "Geräte-Optimierung" mit der tollen Ansage "Increase efficiency by closing apps that run in the background without your knowledge". Ich hoffe, das ist lediglich als Vorschlag gemeint – und beschreibt nicht etwas, was die App ohne Rückfrage selbst tut. Ach ja: Ein Monitor für Zugriffe auf private Daten ist ebenfalls mit dabei, "böse Apps" werden zur Deinstallation gezwungen ("force uninstall"), und obendrein kann man auch die Kontakte sichern und wieder herstellen.

Dem Umfang der App entsprechen auch die wieder einmal nicht begründeten Permissions. Wer alles können will, muss natürlich viel anfordern – doch bei einigen Dingen sollte man dennoch angeben, wozu dies genutzt wird. Da fordern Kaspersky & Co. in ihren Sicherheitsberichten doch explizit auf, man solle nichts installieren, was suspekte Permissions fordert – und dann liefern sie gleich selbst die Beispiele dafür? Wozu möchte beispielsweise *NetQin* Systemeinstellungen oder UI-Einstellungen ändern, Synchronisierungseinstellungen schreiben, Telefonnummern direkt anrufen, sowie lesend und schreibend auf die Kontaktdaten zugreifen? Genau so stellt man sich die Permissions von Malware vor – hier täte eine genauere Beschreibung dringend Not!

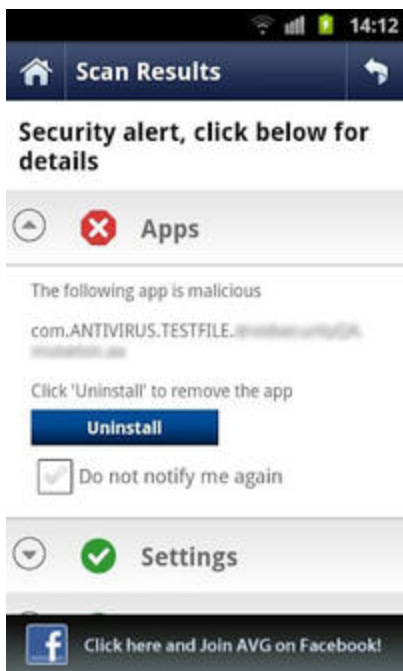
Doch davon einmal ab: Was verspricht *NetQin* denn nun zu leisten? Die Bereiche habe ich ja bereits grob umrissen: Die App kümmert sich um Viren, Malware und Diebstahl-Schutz. Ersteres sowohl in Echtzeit, als auch als "Device-Scan". Für letzteres ermöglicht es die Anzeige der gegenwärtigen Position des Gerätes auf der Karte, eine Sperrung des Gerätes, das Auslösen eines lauten Alarms, und auch das Löschen aller persönlicher Daten. Hinzu kommen noch Tools zum Sichern persönlicher Daten, Memory-Booster, Task-Killer, Traffic-Monitor, File-Manager... Da stellt sich die Frage, was die App eigentlich nicht machen soll. So schön es ist, eine eierlegende Wollmilchsau zu haben – besteht da nicht ein wenig die Gefahr, sich zu verzetteln? Sollte man sich nicht besser auf das Kerngebiet konzentrieren? Doch dieses Urteil überlasse ich besser denen, die die App nutzen...



Anti-Virus und Anti-Malware

Darum geht es in diesem Kapitel ja eigentlich: Malware (nicht verwechseln mit Paintshop & Co.), Viren, und die dazugehörigen Gegenmittel. Mensch holt sich bei Virenbefall (oder vorbeugend) eine Impfung beim Doktor – und Androide greift beispielsweise zu [Dr.Web Anti-virus](#) (rechtes Bild). Endlich mal eine App, die auch bei den Permissions mit gutem Beispiel vorangeht! Zwar wird auch hier nicht erklärt, wozu die App z. B. die "Netzwerkonnektivität ändern" muss – doch damit hat es sich auch bereits: Nur vergleichsweise wenige Permissions werden angefordert, und diese sind (mit genannter Ausnahme) durchaus nachvollziehbar. Zudem sind App und Hersteller bereits aus dem PC-Bereich bekannt, und auch durchschnittliche 4,6 Sterne bei über 25.000 Bewertungen sind geeignet, Vertrauen zu erwecken.

Und was hat der Herr Doktor studiert, was kann er? Die gratis-Version scannt einfach auf "böse Dateien", und sperrt diese in die "Quarantäne". Hierbei scheint sowohl ein Echtzeit-Scan zu erfolgen – als auch die Möglichkeit zu einem "On-Demand-Scan" zu bestehen. Außerdem lässt sich noch einstellen, dass auch die SD-Karte bei jedem Einbinden geprüft werden soll. Die Vollversion bietet dazu auch eine Filterung eingehender Anrufe und SMS, inklusive Blacklist (z. B. für nervige Werbe-Anrufer und Spam-SMS).



Hinter [Anti-Virus Free](#) (linkes Bild) steht die auch vom PC-Anti-Viren-Markt her bekannte Firma AVG. Und bei der App beschränkt man sich schon längst nicht mehr auf das, was der Name suggeriert: Mit an Bord sind auch App-Locker, Task-Killer, Backup, Diebstahlschutz... Kurzum: Auch AVG möchte einen Alleskönner präsentieren.

Was der definitiv kann, ist jede Menge Permissions anzufordern – ohne diese zu erklären. Offensichtlich eine verbreitete Unsitte bei allen Apps in diesem Segment (mit wenigen Ausnahmen): Ich habe einen Namen, ich darf das. Auch hier darf sich der Anwender wieder vergeblich fragen: Wozu bitte will diese App Systemeinstellungen ändern, Synchronisierungseinstellungen schreiben, abonnierte Feeds schreiben, Kontaktdaten lesen und schreiben (da steckt vermutlich das Backup dahinter), das Benutzer-Wörterbuch schreiben (aber nicht lesen?), Kalenderdaten schreiben (aber nicht lesen – hier steht also kaum das Backup als Grund)...

Nun gut: Der Name ist bekannt, die Firma auch. Die Anzahl der Bewertungen liegt im 6-stelligen Bereich, der Schnitt ist über 4 Sternen angesiedelt – alles

Dinge, die geeignet sind, die gerade aufgeworfenen Zweifel ein wenig zu zerstreuen. Schauen wir uns also einmal an, was die App zu können meint:

Es wird ein Echtzeit-Scan angeboten, der auch SMS (huch?) und MMS einschließt. Dazu kommt ein Backup von Kontakten, Anrufprotokollen, Lesezeichen, Text- oder Medianachrichten auf die SD-Karte, Handyortung (durch SMS ausgelöst) sowie "Alarmton" (auch wenn auf lautlos gestellt), Task-Killer, App-Locker (installierte Anwendungen gegen unbefugte Benutzung sperren), sowie der komplette Wipe. Hier als "lokaler Wipe" beschrieben. Keine Erwähnung von Benachrichtigung bei SIM-Wechsel, oder der Möglichkeit, den Wipe remote auszulösen – aber auch diese Funktionalität dürfte enthalten sein.

Besonders schnell will es sein. Einfach und unkompliziert. Besser zu handhaben als *Lookout*, AVG & Kollegen. Die Rede ist von [MyAntiVirus Pro](#) (rechtes Bild). *Full protection for your Android-Phone against Spyware, Virus, Malware, Trojans!* Jawoll! Ob es halten kann, was es verspricht?

Zuerst unser obligatorischer Blick auf die geforderten Permissions. Was? Kann nicht sein – sicherheitshalber direkt im Google Playstore nochmal schauen: Tatsache! Die App will ja fast gar nichts! Scheinbar nur ins Internet, für die täglichen Signatur-Updates. Vorbildlich! Oder, anders ausgedrückt: Wenn diese App es "ganz ohne" kann, wozu brauchen dann die anderen so viel davon? Oder kann die App hier gar nichts, und ist nur eine Demo? Dann wären die gut sieben Euro allerdings eine üble Abzocke. Und nicht einmal 200kB "groß" ist der Download – die anderen bislang betrachteten Apps brachten locker das zehnfache auf die "Waage"...



Was bekommt man nun für genannte sieben Euronen geboten? Laut App-Beschreibung deutschen Support, einen schnellen Anti-Virus-Scan, tägliche Signatur-Updates, sehr geringen Ressourcenverbrauch. Und was meinen die Anwender? Über 300 von ihnen haben die App bewertet, dabei kamen durchschnittlich gut 4,2 Sterne heraus. Die Kommentare lesen sich überwiegend positiv.

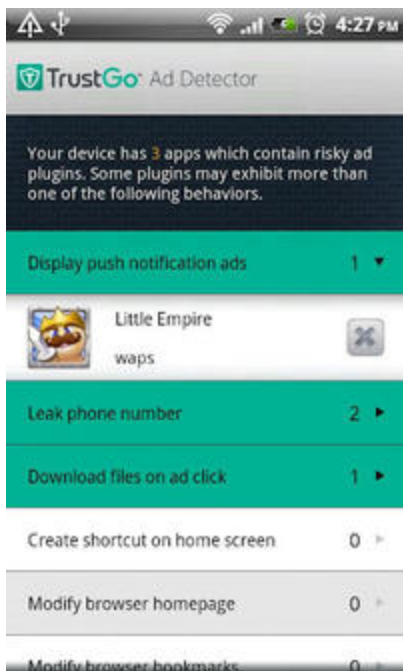
Zusammengefasst ist *MyAntiVirus Pro* eine der wenigen Apps in diesem Bereich, die sich auf ihre eigentlichen Aufgaben konzentriert, somit handlich und leicht bedienbar bleibt, und überdies sparsam im Umgang mit Permissions sowie Ressourcen ist. Schade, dass sich die "Großen" daran kein Beispiel nehmen...

AirPush & Co.

Eigentlich sind das ja alles Werbe-Netzwerke, die da mit Push-Notifications arbeiten. Der bekannteste Vertreter in dieser Gruppe ist sicher [AirPush](#). Warum ich denn Werbung an dieser Stelle behandle? Ganz einfach: Es gibt Werbung (die als solche klar erkennbar zu sein hat) – und es gibt solche, die Huckepack auf einer anderen App mitfährt, sich dann aber getrennt davon und als solche nicht klar erkennbar an ganz anderer Stelle zeigt. Das typische Verhalten eines Trojaners – womit diese Art von Werbung aus meiner Sicht definitiv in die Kategorie "Malware" fällt.



Ein Blick auf das rechte Bild zeigt die Perfidität: Ist das jetzt eine System-Meldung, die mir Tipps für bessere Einstellungen für optimalere Akku-Laufzeit geben will? Mitnichten! Über einen "Klick" auf derartige Einträge hat sich schon mancher Ungewolltes eingehandelt. Das Gemeine daran: Es ist nicht ersichtlich, woher das kommt – etwa welche App dafür verantwortlich ist. Fein raus ist da, wer bereits Jelly Bean auf seinem Androiden hat: Ein mit dieser Android-Version (4.1) eingeführtes neues Feature erlaubt, durch langes Drücken auf eine Notification den "Übeltäter" zu entlarven – und ihm das "Übeltun" zu untersagen. Und was machen die anderen 95% der Android-Anwender?

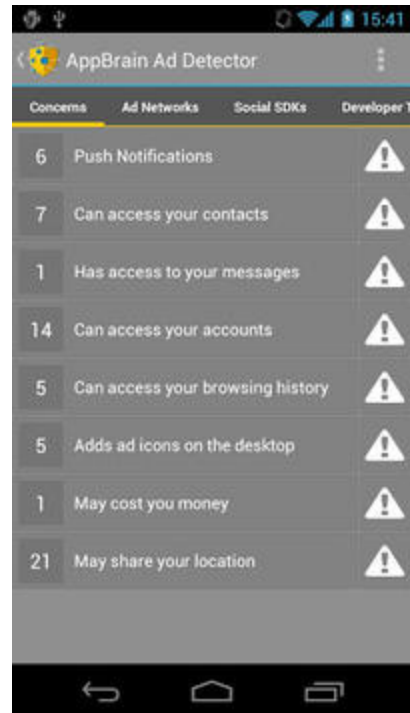


Natürlich muss niemand in die Röhre gucken. Recht bald nach Auftauchen dieser Schädlinge fand sich auch die erste Software, die diese aufspüren konnte. Recht früh mit dabei, und nach wie vor auch in der Bewertung an der Spitze ist [AirPush Detector](#). Und obwohl sich der Begriff "AirPush" als Bezeichnung dieser Art von unlauterer Werbung etabliert hat, gibt es noch weitere Netzwerke dieser Art – weshalb in Folge auch einige Apps auftauchten, die möglichst viele dieser Übeltäter aufspüren wollten. Ein Beispiel dafür ist der links abgebildete [Ad-Detektor](#) von TrustGo. Wie im Screenshot bereits erkennbar, spürt diese App nicht nur AirPush-Trojaner auf: *TrustGo Ad Detector scannt und schützt Ihr Mobiltelefon vor möglichen Verletzungen der Privatsphäre und die Identität Lecks durch Anzeigen von Anwendungen über die am häufigsten verwendeten Werbe-Netzwerke angezeigt.* (aus der App-Beschreibung).

Damit man aber nicht erst warten muss, bis man sich bereits "infiziert" hat, gibt es auch Vorsorge-Apps. So lässt sich etwa mit dem [AppBrain Ad-Detector](#) schon vor der Installation feststellen, welche App eventuell versucht ist (Bild rechts). Passenderweise wird das zusammen mit den verlangten Berechtigungen angezeigt. Wer ohnehin bereits auf die [AppBrain Market](#) App als gute und schnelle Alternative zur überladenen Playstore-App setzt, kann diesen Ad-Detector dort integrieren – so ist dann alles passend zusammengefasst: Neue App gesucht, direkt geprüft, sofort entsorgt – und eine saubere Alternative gefunden sowie installiert...

Lange genug hat dieser Ärger angehalten – doch endlich hat sich auch Google des Themas angenommen, und [die Regeln im Market verschärft](#). Diese Art von schlechter Werbung sollte es also in Zukunft schwerer haben. Warum ich das Thema dann hier dennoch so ausführlich behandelt habe? So mancher nutzt halt auch alternative Märkte. Und ob diese allesamt ebenfalls der Trojanischen Werbung den Kampf ansagen, ist weniger sicher...

Natürlich gibt es auch zu diesem Thema wieder eine passende [Übersicht bei AndroidPIT](#), in der sich weitere Alternativen und Tipps finden. Wie beispielsweise der, zum Abschalten der Airpush-Werbung so ganz ohne App das [Opt-Out auf der AirPush-Website](#) zu nutzen – natürlich nur, sofern man auch betroffen ist.

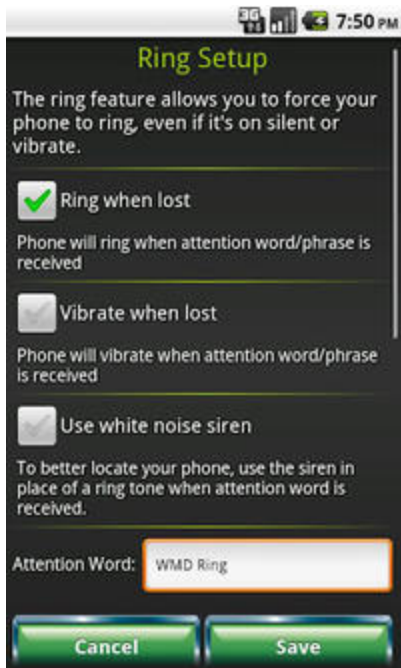


Diebstahlschutz

Kann eine App wirklich vor Diebstahl schützen? Sicher nur bedingt. Dafür müsste sie den Dieb "in flagranti außer Gefecht setzen": Stromstoß, Tränen- und Nervengas, Keule auf die Rübe... Menschenrechts-Kommission vor der Tür. Geht so nicht. Abgesehen von der schwierigen technischen Umsetzung.

Dass das doch zumindest ansatzweise geht, zeigt u. a. [Thief Alarm](#) (rechtes Bild): Hier soll der Dieb mit Sirenengeheul in die Flucht geschlagen werden. Der Krawall geht dabei los, sobald man das Gerät bewegt, ohne rechtzeitig den Alarm-Code einzugeben. Eine Sache, die sich sicher auch [mit Tasker umsetzen](#) ließe – eventuell sogar mit einer Gesten-Deaktivierung (einmal kurz schütteln, einmal drehen: Deaktiviert)...





Die meisten Anti-Theft Apps sind eher die "Pille danach" – sie greifen erst dann ein, wenn das Kind in den Brunnen gefallen und der Androide verschwunden ist. Die dann angebotenen Möglichkeiten fallen sehr unterschiedlich aus; vereinfacht ausgedrückt, reichen sie von einfachen Lokalisierungshilfen bis hin zur Selbstzerstörungs-Automatik. Eindeutig in die erste Kategorie gehört das links abgebildete [Wheres My Droid](#): Ist der Droide verschwunden, benötigt man eben mal sein Zweitgerät, das Telefon eines Freundes, oder einer beliebigen anderen Person. Damit schickt man sodann eine SMS mit dem "Attention Word" (also dem passenden Code, den man zuvor konfiguriert hat) an das verschwundene Gerät – und es antwortet per SMS mit den Koordinaten. Ein Blick in Google Maps – aha, da isses (gerade)! Ein anderes "Attention Word", und das Gerät macht mit ordentlich Krach auf sich aufmerksam. Auch wenn es gerade "ruhig gestellt" (silent mode) war.

Das setzt natürlich voraus, dass die eigene SIM-Karte noch im Gerät steckt (sonst läuft die SMS ja ins Leere). Aber per EMail klappt es auch, und das sollte nach SIM-Wechsel auch noch tun. Bei selbigem benachrichtigt diese App allerdings nicht – auch ein "Wipe" persönlicher Daten ist hier nicht vorgesehen.

[WatchDroid Lite](#) kann dies auch, bietet aber zusätzlich einen "Stealth Mode" (Tarn-Modus), damit der "Finder" die App nicht einfach löscht. Krach schlagen und SMS mit GPS-Daten verschicken sollte damit kein Problem sein. [WatchDroid Pro](#) (rechtes Bild) bietet für etwa anderthalb Euro einen guten Mehrwert dazu: Das Telefon lässt sich nun auch aus der Ferne sperren, oder komplett löschen (Wipe). Auch ein eventueller SIM-Karten-Wechsel wird von der Pro-Version erkannt – und mit dem Versand einer SMS an den hinterlegten Empfänger beantwortet. Der Trend geht eindeutig zum Zweitgerät...





Mehr als nur eben interessant klingt auch der Ansatz von [GotYa! Face trap](#) (rechtes Bild) – zumindest wenn der Androide über eine Front-Kamera verfügt. Bei jeder Falscheingabe des Entsperrcodes/Entsperrmusters schickt diese App ein Foto des "Täters" inklusive einem Google-Maps-Link seiner gegenwärtigen Position per Mail. Per SMS lässt sich aus der Ferne auch jederzeit der gegenwärtige Standort abfragen, ein Rückruf auslösen, oder ein Alarm abspielen (falls man das Gerät lediglich verlegt hat, und es auf die letzten Meter "akustisch lokalisieren" möchte). Die Vollversion benachrichtigt auch bei Wechsel der SIM-Karte.

Etwa zwei Euro kostet der "Spaß", eine Gratis-Version zum Testen gibt es jedoch ebenfalls.

Dann wären da natürlich noch die als [Rundum-Sorglos-Pakete](#) aufgeführten Apps, die sich ja "mal so eben nebenbei" auch noch mit um den Diebstahl-Schutz kümmern wollten. Und natürlich gibt es auch hier wieder eine [Übersicht bei AndroidPIT](#), die noch eine ganze Reihe weiterer Apps zum Thema kennt.

Wie beispielsweise das auch recht interessant klingende Tool [TotalCare - Remote Security](#). Dieses ermöglicht es (siehe Screenshot rechts), vom verlorenen Gerät noch die wichtigsten Daten zu kopieren – bevor man ihm den Garaus in Form einer Löschung macht. Eine gute Sache, so beispielsweise das letzte Backup schon ein wenig zurückliegt... Natürlich lässt sich auch der gegenwärtige Standort des Gerätes feststellen, ebenso erfolgt eine Benachrichtigung bei Wechsel der SIM-Karte.



App-Sicherheit

Nachdem nun etliche Sicherheits-Apps besprochen wurden, wird es Zeit, das Wort einmal umzudrehen: Wie steht es um die App-Sicherheit? Denn was nützen all die tollen Sicherheits-Apps, wenn die Anwendungen selbst unsicher sind? Das beste Sicherheits-Schloss ist sinnlos, wenn man den Schlüssel auf dem Fensterbrett ablegt, und dann auch noch das Fenster offen stehen lässt.

In diesem Kontext gilt es, mehrere Bereiche zu betrachten: Zum Einen geht es um die auf dem Gerät gespeicherten Daten – und zum Anderen sind da die Daten, die über diverse Netzwerke übertragen werden. Werfen wir also einmal einen Blick auf beide Themen.

Gespeicherte Daten

Es soll ja vorkommen, dass jemand "zufällig" ein Android-Gerät "findet" (manchmal auch in der Tasche der Jacke, die der Eigentümer gerade trägt). Dieser "Finder" kann nun lediglich daran interessiert sein, die "gefundene" Hardware zu Geld zu machen – oder aber auch, die darauf befindlichen Daten zu nutzen. Besonders interessant sind dabei natürlich Benutzernamen und Passwörter, aber auch andere persönliche Daten – denn beide lassen sich vorzüglich zum Identitäts-Diebstahl nutzen. Um "im Namen des Eigentümers" Einkäufe zu tätigen, oder Spam zu versenden, um nur zwei Beispiele zu nennen. Daher sollten diese Daten – wenn überhaupt – auf sichere Weise auf dem Gerät gespeichert werden.

Wie so etwas sicher geschehen kann, dafür gibt es mehrere Ansätze, die zumeist etwas mit Verschlüsselungs-Techniken zu tun haben. Absolut klar sollte jedoch sein: Passwörter im Klartext (also ohne zusätzliche Hilfsmittel lesbar) auf dem Gerät zu speichern, ist ein absolutes No-Go – selbst die Verwendung einer einfachen "Cäsar-Chiffre" (siehe [Texte verschlüsseln](#)) bietet da schon wesentlich höhere Sicherheit, auch wenn sie alles andere als sicher ist: Auch "Daten-Diebe" sind von Natur aus faul; Klartext-Passwörter sind ohne großen Aufwand nutzbar, für alles andere muss zunächst der Algorithmus ermittelt werden.

Wer jetzt meint, das wäre doch selbstverständlich – der lese bitte weiter. Denn offensichtlich ist dem nicht so. Zur Ermittlung einiger Schwachstellen habe ich ein [Nandroid-Backup](#) meines Motorola Milestone 2 hergenommen, welches ich gleich nach dem Rooten und der Installation eines [Custom Recovery Images](#) erstellt habe (Stock Android 2.2.2). Daraus habe ich die /data Partition extrahiert, und bin die Verzeichnisse durchgegangen. Da gibt es erstaunliches zu entdecken...

System-Daten

Beginnen wir bei "des Pudels Kern": Dem Android System. Dies sollte ja gerade in diesem Punkt ein großes Vorbild sein – was auch weitgehend zutrifft. Finden sich hier etwa sensible Daten im Klartext? Leider ja.

Ein Beispiel, das ich hier aufführen möchte, ist die Datei /data/misc/wifi/wpa_supplicant.conf. Hier werden die konfigurierten WLAN Accesspoints gespeichert. Ein Ausschnitt dieser Datei sieht etwa wie folgt aus:

```
network={
    ssid="platzhotel"
```



```
    psk="hotel123"  
    key_mgmt=WPA-PSK  
    priority=46  
}
```

Da gibt es also einen WLAN-Accesspoint namens "platzhotel", das mit WPA verschlüsselt ist. Der zugehörige Schlüssel lautet: "hotel123". Bummer! Ja, tatsächlich: Alles im Klartext (Anmerkung: Die Daten habe ich natürlich leicht angepasst, um niemanden in Bedrängnis zu bringen). Hier lässt der Name des Netzes zudem noch leicht auf die Institution schließen: Es handelt sich offensichtlich um ein Hotel, welches den Namen "Platzhotel" trägt. Ein deutscher Name, was die Location zusätzlich eingrenzt. Jetzt noch schnell die App [Location Cache](#) installiert und geschaut, wo der Eigentümer zuletzt unterwegs war (WLANs werden hier ja separat ausgewiesen) – und mit etwas Glück findet man auch den passenden Platz.

Im Falle des WLANs eines Hotels oder Cafés ist dies vielleicht noch kein "großes Ding", da die Betreiber ihren Gästen den Zugang meist auf simple Anfrage mitteilen. Die genannte Datei speichert aber auch die Zugangsdaten für sensiblere Firmen-Netzwerke, so genutzt – und spätestens hier wird so etwas zum Sicherheits-Risiko!

Daten von Hersteller-Apps

Hersteller und Netzbetreiber "segnen" uns ja mit weiteren vorinstallierten Apps. Manche davon sind auch wirklich für die breite Masse nützlich – andere möchte der Anwender viel lieber loswerden (siehe [Bloatware](#)). In die erste Kategorie gehört u.a. auch eine App zum Empfang, Lesen, und Versand von Mails – die sich möglichst mit vielen Anbietern verträgt.

Hierfür sorgt u. a. auch HTC bei seinen Geräten, und die App ist recht beliebt. Insbesondere auch bei denjenigen, die etwa auf ihre Firmen-Mails auf dem Exchange-Server zugreifen möchten. Und hier lauert er wieder, unser "schwerwiegender Systemfehler" namens "Zugangsdaten im Klartext": Account, Passwort, und sogar Domain-Daten lassen sich so direkt einsehen! Wie das an einem Beispiel aussieht, darauf geht die "Franzis-Edition" dieses Buches ein.

App-Daten

Wenn sich schon beim System solche Lücken finden, überrascht es sicher kaum, dass so etwas auch bei Apps auftritt. Ein wirklich extremes Beispiel liefert etwa die populäre App Skype, wie ein [Artikel bei AndroidPolice.COM](#) ausführlich beschreibt.

Nicht nur speichert die App den Benutzernamen, Profildaten, Chat-Protokolle, und mehr unverschlüsselt in diversen SQLite-Datenbanken in seinem Daten-Verzeichnis (unterhalb von /data/data/com.skype.merlin_mecha/) – sondern sie hebt auch gleich noch sämtliche vom System vorgesehene Sicherheitsmaßnahmen aus. Eigentlich haben nämlich nur die App selbst und natürlich root Zugriff auf diese Verzeichnisse. Skype legt dies jedoch anders fest, und macht sie für "alle Welt" lesbar und schreibbar! Beispiele aus dem verlinkten Artikel:

```
# ls -l /data/data/com.skype.merlin_mecha/files/shared.xml  
-rw-rw-rw- app_152 app_152 56136 2011-04-13 00:07 shared.xml
```

```
# grep Default /data/data/com.skype.merlin_mecha/files/shared.xml
<Default>jcaseap</Default>
```

Damit lässt sich der Name des Anwenders ermitteln – und somit auch der Name des Verzeichnisses, in dem seine Daten gespeichert wurden. Aber auch ohne diesen Schritt wäre es zu finden gewesen: So viele Verzeichnisse gibt es hier nicht, und alles steht ja offen wie ein Scheunentor. Werfen wir also einen Blick auf die Daten dieses Benutzers:

```
# ls -l /data/data/com.skype.merlin_mecha/files/jcaseap
-rw-rw-rw- app_152 app_152 331776 2011-04-13 00:08 main.db
-rw-rw-rw- app_152 app_152 119528 2011-04-13 00:08 main.db-journal
-rw-rw-rw- app_152 app_152 40960 2011-04-11 14:05 keyval.db
-rw-rw-rw- app_152 app_152 3522 2011-04-12 23:39 config.xml
drwxrwxrwx app_152 app_152 2011-04-11 14:05 voicemail
-rw-rw-rw- app_152 app_152 0 2011-04-11 14:05 config.lck
-rw-rw-rw- app_152 app_152 61440 2011-04-13 00:08 bistats.db
drwxrwxrwx app_152 app_152 2011-04-12 21:49 chatsync
-rw-rw-rw- app_152 app_152 12824 2011-04-11 14:05 keyval.db-journal
-rw-rw-rw- app_152 app_152 33344 2011-04-13 00:08 bistats.db-journal
```

Hier gibt es also eine SQLite-Datenbank mit dem Namen main.db. Klingt verlockend. Und lässt sich mit einem beliebigen SQLite-Client öffnen – es darf ja jeder zugreifen, Dank der tollen Zugriffsrechte. In dieser Datenbank findet sich u. a. eine Tabelle namens accounts – wenn das nicht interessant klingt! Und was da alles schönes drin gespeichert ist: Kontostand, vollständiger Name, Geburtsdatum, Anschrift, Telefon, Mail-Adresse, Website, Biografie, und mehr – kurzum alles, was man seinem Skype-Account anvertraut hat...

Übertragene Daten

Auch hier gilt wieder: Wer bei Problemen mit unverschlüsselt übertragenen Daten nur an Apps von "irgendwelchen kleinen Drittanbietern" denkt: Fehlanzeige. Bis etwa Mitte 2011 taten sich hier insbesondere [Facebook und sogar Google](#) hervor, die zur Datenübertragung statt auf sicheres [HTTPS](#) auf unverschlüsseltes [HTTP](#) setzten – und zwar auch für Anmeldedaten. War das Gerät dann auch noch in einem "offenen WLAN" eingebucht, war es für potentielle Angreifer ein einfaches, diese Zugangsdaten abzugreifen – indem sie sich im selben WLAN anmeldeten, und beispielsweise einen [Sniffer](#) einsetzten.

Zumindest haben die beiden Genannten diese Sicherheitslücke geschlossen. Heißt es nun also "generelles Aufatmen"? Keineswegs! Denn dieses Problem taucht leider immer wieder auf. Aktuellstes Beispiel ist die beliebte App [WhatsApp Messenger](#), die [bis vor kurzem das gleiche tat](#). Auch hier wurde die Lücke zum Glück [bereits geschossen](#). Doch wer weiß, wo sie sich als nächstes zeigt? Auch wenn die Beispiele deutlich machen, dass insbesondere Apps sozialer Netzwerke dabei gern in den Vordergrund geraten – das zeigt lediglich, dass es hier besonders auffällt. Es bedeutet jedoch nicht, dass sich das Phänomen darauf beschränkt.

Auch diese Behauptung möchte ich nicht einfach so im Raum stehen lassen. So hat AndroidPIT-Mitglied [Jörg Voß](#) die Nachrichten-App [N24](#) genauer [unter die Lupe genommen](#). Hatten deren Entwickler doch versprochen, nicht auf sensible Daten wie insbesondere die [IMEI](#) zuzugreifen – da diese den Benutzer eindeutig identifizierbar machen würde. Gewissermaßen wurde auch Wort gehalten – doch

stattdessen verwendet man nun die Geräte-ID, die ebenso eindeutig ist. Und überträgt sie natürlich auch noch unverschlüsselt über das Netz. Als ob dies nicht ausreichen würde, ist Ziel der Übertragung auch noch ein Drittanbieter. Noch nicht schlimm genug? Es geht schlimmer: Bei diesem Drittanbieter handelt es sich um ein Werbe-Netzwerk (der Ziel-Server: mobile.smartadserver.com in Frankreich).

Aber wie kann man sich vor derartigen Problemen schützen? [GMV](#) allein hilft hier offensichtlich wenig: Man sieht so etwas einer App ja wohl kaum "von Außen" an. Und in den App-Beschreibungen wird sich kaum ein Satz finden wie "Wir übertragen ihre persönlichen Informationen zu Werbezwecken unverschlüsselt an Dritte". Man könnte jetzt ständig sämtliche Blogs und News-Feeds verfolgen (mal ehrlich, wer macht so etwas – außer mir natürlich?), doch derartige Sicherheitslücken sind ja nicht bereits bei Erscheinen einer App bekannt.

Ein wichtiger Punkt dürfte definitiv deutlich geworden sein: Im Umgang mit offenen (und öffentlichen) Netzen besonders vorsichtig zu sein; erstere am Besten weitgehend meiden. Das schränkt das Risiko schon ein ganzes Stück ein.

Aber auch Andere haben dieses Problem erkannt, und bieten ihre Unterstützung an. So ist die Website des Services [AppWatchDog](#) der Forensik-Firma [viaForensics](#) definitiv einen Besuch wert. Hier nimmt man bekannte Apps genauestens unter die Lupe – und es ist nicht gerade leicht für eine App, die strengen Tests mit einem grünen Haken zu bestehen. Es ist eher erschreckend, wie viele Apps hier durchfallen: Schränkt man die Anzeige auf "Android" und "Productivity" ein, findet sich etwa kein einziger grüner Haken! Zwar sieht es für das iPhone nicht wesentlich besser aus (peinlicherweise ist der einzige grüne Haken hier bei "Google Mail" gesetzt – da fragt man sich ernsthaft, warum die App das bei Android nicht geschafft hat?) – das zeigt jedoch lediglich, dass es im Bereich mobiler Apps um die Sicherheit nicht sonderlich gut bestellt ist. Ein Blick auf die Details entschärft den Schreck häufig ein wenig, wie beispielsweise der Eintrag zu [K-9 Mail](#) zeigt: Keine unverschlüsselt gespeicherten Passwörter – jedoch werden Benutzernamen und Daten unverschlüsselt abgelegt.

Bei AppWatchDog werden die Apps in Intervallen erneut geprüft – man sieht also, ob die Entwickler die Warnung ernst genommen haben. Erfreulich zu sehen, dass sich da bei einigen Apps tatsächlich etwas tut! Und übrigens: Wem eine App besonders am Herzen liegt, der kann auch deren Prüfung auf der genannten Webseite anregen.

Weitere Sicherheits-Probleme

Als ob obiges noch nicht genügt, bietet die Sicherheit noch einige andere Tücken – auf die ich hier nicht in voller Breite eingehen möchte. Ein paar Beispiele seien jedoch noch genannt:

Um die Authentizität von Apps zu gewährleisten, muss jede im Playstore eingestellte App mit einem Zertifikat signiert werden, welches den Entwickler eindeutig identifiziert. Dies erschwert es u. a. Betrügern, eine schädliche App als "Update" einer beliebten anderen App auszugeben. Was aber, wenn ein solches Zertifikat einmal kompromittiert wurde – also "in fremde Hände" geriet? Das mag zwar dem einen oder anderen Entwickler "peinlich" sein – ein guter Zug ist es jedoch, die Anwender darauf unverzüglich hinzuweisen (etwa in der App-Beschreibung). Auf jeden Fall ist dann ein neues Zertifikat fällig.

Die Sache hat jedoch einen kleinen Haken: Neues Zertifikat heißt auch, dass Updates auf die mit dem alten Zertifikat signierten Versionen nicht mehr möglich sind (sonst wäre das System ja ad absurdum geführt). Besonders ärgerlich für die Anwender, die bei einem Update ihre Daten nicht mehr übernehmen können! Hier könnten Entwickler entsprechend Vorsorge treffen, indem sie – auch in den gratis-Versionen ihrer Apps – einen Datenexport ermöglichen. Gibt es zu der App auch eine Kaufversion, könnten auf diese Weise auch dort die Daten dann übernommen werden. Ebenso wäre dies bei einer neuen Version mit neuem Zertifikat möglich.

Wer bis hierher mitgelesen hat könnte nun denken: "Aha, wenn ich also auf Apps von Drittanbietern verzichte..." Leider nicht ganz. Zum Einen haben obige Beispiele ja gezeigt, dass durchaus auch vorinstallierte Apps betroffen sein können – und zum Anderen macht Android so ganz ohne Apps ja auch nicht wirklich Spaß. Wer derart paranoid ist, sollte dann besser zu einem der "guten alten Knochen" greifen, die lediglich Telefonie und SMS unterstützen.

Und dummerweise gibt es auch ohne Apps hin und wieder Sicherheitsprobleme – etwa mit proprietärer Software vom originalen Hersteller. So [berichtet etwa Android-TV](#) von einer Sicherheitslücke in der Software "Kies", die Samsung für seine Geräte einsetzt (für die Daten-Synchronisation sowie auch für Firmware-Updates). Hier zeigte sich allerdings das vorbildliche Verhalten des Herstellers: Kaum war die Lücke bekannt, schon war sie wieder geschlossen. Und Hand aufs Herz: Vor Fehlern gefeit ist niemand – so etwas kann also wirklich jedem Entwickler passieren. Gut, wenn dieser dann so schnell reagiert, wie in diesem Falle Samsung!

Wen das Thema noch näher interessiert, der findet im Netz natürlich weitere detaillierte Informationen. Etwa in einem sehr ausführlichen Artikel der Computerwoche zum Thema [Android Sicherheitskonzept](#) – oder in einem Block bei AndroidPIT mit dem Titel [Android muss sicher werden](#). Eine weitere empfehlenswerte Quelle ist das Buch *Android Forensics and mobile Security* von Andrew Hoog, das voraussichtlich im Oktober 2012 in einer deutschen Übersetzung im Franzis-Verlag erscheinen wird.

SYSTEM

Im zweiten Teil dieses Buches soll es darum gehen, einen Überblick über das Android-System zu erhalten – und es in den Griff zu bekommen. Die folgenden Kapitel sollen daher Antworten auf die Frage "Wo finde ich was?" liefern – und ebenso aufzeigen, wie sich Dinge an den eigenen Bedarf anpassen lassen.

Der Super-User "root"

Von ihm ist in diesem Buch häufiger einmal die Rede: Dem "Wurzel-Account" (engl. "root" = "Wurzel"), ohne den das Eine oder Andere wieder einmal nicht geht. Und da es sich dabei um den zentralen System-Account handelt, soll er in diesem Kapitel auch gleich als Erstes behandelt werden.

Für die Hersteller/Provider scheint es sich hier um "die Wurzel allen Übels" zu handeln: Dieser Account hat so viel Power, dass man den Anwender da besser gar nicht erst heran lässt. Zu groß die Gefahren, dass er sich kurze Zeit später mit "Garantieansprüchen" meldet, weil er sich damit "etwas kaputt gemacht" hat. Die einzige mir bekannte Ausnahme betrifft den US-Provider *Cincinatti Bell*, der seine Kunden nicht länger auf ein Motorola-Update warten lassen wollte – und daher schlicht eines selbst erstellte. Damit die Kunden dies auch installieren konnten, wurde die Anleitung zum Rooten gleich mitgeliefert – bei voller Garantie durch den Provider selbst. Eine rühmliche Ausnahme... Die aber noch immer nicht erklärt, um was es sich bei "root" eigentlich handelt.

Kauft man einen Windows-PC, gibt es auf diesem einen Account für den Benutzer "Administrator" – dem man bei der Ersteinrichtung ein Passwort verpasst. Installiert man Linux, heißt das Pendant "root" (bei einem Mac sicher ähnlich). Android basiert auf Linux – aber trotzdem gönnen uns die Hersteller den root-Zugang in der Regel nicht, sondern drohen: "Wer sich root-Zugang zu seinem Gerät verschafft, verwirkt damit den Garantieanspruch."

Damit ist nun klar, um was es bei dem Wort "root" geht: Um den administrativen Zugang zum System, mit dem man alles (kaputt) machen kann. Naja, fast alles – die Hardware wohl eher nicht. Weshalb die Warnung mit der Garantie wohl letztendlich vor Gericht kaum haltbar sein dürfte, wenn man z. B. das Display wechseln lassen muss, oder der interne Speicher den Geist aufgibt (anders sieht es aus, wenn die CPU verglüht, weil man sie hoffnungslos übertaktet hat – siehe [CPU übertakten](#)).

Braucht man den root-Zugang denn nun wirklich? Ja und nein. Wer mit seinem Gerät, dessen Funktionen, sowie der verwendeten Software bereits rundum zufrieden ist, alles so läuft, wie gewünscht, und "eigentlich" nichts vermisst – der braucht auch keinen root-Zugang. Er hat ja bereits alles, was er braucht. Hat man hingegen ein Problem, was sich ohne den root-Zugang nicht lösen lässt, sieht das schon anders aus: Je nachdem, wie schwer es einen trifft, neigt sich das Zünglein an der Waage immer mehr der Anzeige zu, die mit "mach mich root!" beschriftet ist.

Um die Entscheidung zu erleichtern, schauen wir uns zuerst einmal die Vor- und Nachteile an, die mit dem Rooten verbunden sind:

Vorteile des root-Zugangs

Eine ganze Reihe von Apps setzen ein gerootetes Gerät voraus. Oder stellen auf einem solchen zusätzliche Funktionen bereit. Zahlreiche Einstellungen und Änderungen lassen sich ohne root-Zugang gar nicht vornehmen:

- [Übertakten](#) bzw. [Untertakten](#) der CPU – um entweder mehr Leistung herauszukitzeln, oder den Akku zu längerem Durchhalten zu animieren

- [Entfernen/Einfrieren vorinstallierter Apps](#) – da kommen oftmals Dinge mit, die kein Mensch braucht. Ab Android 4.0 (a. k. a. *Ice Cream Sandwich*) lassen sich unerwünschte Kandidaten auch ohne root zumindest "einfrieren", also deaktivieren.
- [Bearbeiten der Start-Events](#) und somit verhindern, dass unerwünschte Apps automatisch gestartet werden. Zwar gibt es auch Apps, die versprechen, das ohne root zu erledigen – doch diese schießen die betreffenden Apps lediglich nach dem Start wieder aus dem Speicher. Worauf sich die jeweiligen Apps meist einfach wieder starten. Mit root kann man den Start selbst verhindern.
- [Neu-Kalibrieren des Akkus](#) – eigentlich eher dem Android-System die Akku-Daten neu beibringen, indem veraltete Akku-Statistiken gelöscht werden. Als Folge davon ist die Anzeige der verbleibenden Kapazität/ Laufzeit wieder genauer.
- [Optimierung der Speicherverwaltung](#) – personalisierte Einstellungen statt 08-15, damit der Droide wieder flüssiger läuft
- [Anlegen einer Swap-Datei](#) und somit Nutzung von "virtuellem RAM"
- [App2SD auch mit Android < 2.2](#), bzw. die Nutzung erweiterter Varianten wie App2SD+/Link2SD
- Aufspielen alternativer Firmware (AKA "[Custom ROM](#)")
- Ändern der Systemschriftart(en)
- Erstellen eines wirklich [vollständigen Backups](#) des Android-Systems (siehe auch [Nandroid Backup](#))
- Einrichten einer [Firewall](#)
- Einfaches erstellen von Screenshots, ohne viel Verrenkungen (ja, auf Samsung GalaxyS geht das von Haus aus – und Android 4.0 bringt das auf alle Geräte)

Diese Liste ist keinesfalls vollständig (natürlich auch nicht nach Relevanz sortiert – die wäre ohnehin wieder sehr subjektiv). Mit root hat man quasi überall Zugang – keine Ecke des Android-Systems bleibt verschlossen. Genau da liegt auch das Risiko – aber da liegt es auch beim root-Zugang auf dem Linux PC, oder dem Administrator-Zugang beim Windows-PC:

Risiken des root-Zugangs

Die Risiken sind schnell mit einem Satz beschrieben: Falsch angewendet, kann man sich mit root-Zugang das System unbrauchbar machen. Im schlimmsten Fall verwandelt man gar seinen Androiden in einen Ziegelstein – wenn man z. B. ohne Sinn und Verstand die CPU hoffnungslos übertaktet, und diese schließlich den Hitzetod stirbt. Mit Wissen und Verstand eingesetzt, ist der root-Zugang ein mächtiges und nützliches Werkzeug. Quasi wie ein Autoschlüssel: Setzt sich der 8-jährige Steppke damit hinters Steuer... Womit wieder bewiesen ist, dass man uns für absolut unmündig hält...

Oder sich schlicht vor unnötigen Rückgaben und Garantie-Einforderungen von sich selbst überschätzenden Anwendern zu schützen. Ein nachvollziehbarer Grund – denn solche Anwender gibt es leider zu viele...

Wie bekomme ich root-Zugang?

Das jetzt so zu erklären, dass es für jeden gilt, führt ein wenig zu weit. Für diese Übersicht kurz zusammengefasst, gibt es da mehrere Möglichkeiten – und je nachdem, um welches Gerät es geht, greift davon eine, keine, oder mehrere.

Da ist zum einen "Software-root": Man lädt sich die passende App auf den Androiden, startet sie, und bestätigt: "Ja, ich will root!". Fertig. Toll: Mit so einem Gerät fühle ich mich absolut sicher. Wer sagt mir, dass eine andere App das nicht im Hintergrund tut, ohne mich zu fragen? Die meisten dieser Apps haben allerdings mit Gingerbread ohnehin ihren Dienst eingestellt.

OK, auch die zweite Variante ist im Prinzip eine Art "Software-root" (schließlich geht es ja um Software-seitigen Zugang). Nur geht es hier nicht um eine "einfache App", sondern es ist schwieriger: Zunächst muss das USB-Debugging im Gerät aktiviert werden (expliziter Schritt, schwer von einer App auszuführen). Dann ist der Androide per USB-Kabel mit dem PC zu verbinden (unmöglich, dass das eine App im Hintergrund macht). Und schließlich muss man auf dem PC die "root-Software" starten, die über das Kabel auf das Android-Gerät zugreift. Die Schritte sind noch immer einfach und nachvollziehbar – aber hier habe ich keine Bedenken, dass das ohne mein Zutun passieren könnte. Leider sind die meisten "Allrounder" aus dieser Kategorie (wie etwa das bekannte [SuperOneClick](#)) auf ein auf dem PC laufendes Windows angewiesen – womit Mac OS- und Linux-Anwender im Regen stehen bleiben.

Welche Variante jetzt für ein bestimmtes Gerät verfügbar ist, und welche Software dafür benötigt wird, recherchiert man am besten im Forum. Bei AndroidPIT gibt es gerätespezifische Foren (z. B. eines für das [HTC Wildfire](#), eines für das [HTC Desire](#), für das [Motorola Milestone](#), und so weiter). Jedes dieser Foren hat ein Unter-Forum für root-Fragen – dort finden sich die Informationen, die für das jeweilige Gerät zutreffend sind. Auch ein Blick in den [root Artikel des AndroidPIT Wikis](#) kann sich für weitere Informationen als nützlich erweisen.

Laufen dann alle Apps mit root-Rechten?

Eine oft aufkommende Befürchtung – zum Glück unbegründet. Also die kurze Antwort: Nein, nicht ohne ausdrücklichen Wunsch des Anwenders.

Für eine detaillierte Antwort muss ich etwas tiefer greifen. Und wir müssen uns in Erinnerung rufen: Ein Android-System läuft ja mit Linux, also gelten hier auch entsprechende Richtlinien. Und jede App läuft darüber hinaus unter einem eigenen Benutzer. Auch root ist ein Benutzer, wenn auch ein ganz spezieller. Und wenn eine "normale App" etwas mit root-Rechten ausführen möchte, muss sie "root" dazu auffordern. Der Befehl dazu heißt [sudo](#), was wir in unserem speziellen Kontext mit "SuperUser, DO ..." wiedergeben können.

Läuft sie jedoch nicht unter "root" (und das ist bei Android die Regel: Jede App läuft, wie bereits gesagt,



unter einem eigenen User), dann muss sie für Aktionen, die root-Rechte benötigen, root halt höflich bitten – und das tut sie, indem sie dem auszuführenden Befehl ein "sudo" (unter Linux; bei Android heißt der Befehl dazu schlicht "su") voranstellt. Also "sudo <Befehl>". Derart geweckt, schaut der SuperUser in seiner "Datenbank" nach, ob die App denn so etwas darf. Beim ersten Aufruf steht sie da noch nicht drin: Die Folge ist ein Popup der SuperUser-App "App xyz möchte etwas mit SuperUser-Rechten machen. Darf sie das?". Dazu zwei Buttons für "Ja" und "Nein", sowie eine "Checkbox", ob sich SuperUser diese Entscheidung für die Zukunft merken soll.

Bei jedem weiteren Aufruf findet der SuperUser die App in seiner Datenbank mit dem Vermerk "die darf das immer", und führt den Befehl direkt aus. Zur Sicherheit wird dieser Fakt jetzt nochmals als Hinweis eingeblendet (siehe Screenshot rechts). Die App wird dabei nicht gebremst, es ist auch keine Interaktion nötig. Daher sollte das in diesem Falle dann sogar vom Lockscreen aus funktionieren. Etwas störend ist das natürlich im Falle einer Screenshot-App, wie das Bild zeigt – da dieser Hinweis dann auf jedem Bild verewigt ist. Deshalb lässt er sich auch in den Einstellungen der SuperUser-App abschalten.

Gibt es irgendwo weitere Info zu diesem Thema?

Wer noch unsicher ist, findet im Netz natürlich noch weitere Informationsquellen – überwiegend in Foren und Wikis. Eine kleine Auswahl sei hier kurz aufgeführt:

- AndroidPIT Forum: [Wozu Rooten/Flashen?](#)
- AndroidPIT Blog: [Rooten leicht gemacht](#) (11/2011)
- Brutzelstube: [Rooten – und Garantie?](#)
In der Brutzelstube gibt es noch zahlreiche weitere Informationen, insbesondere – aber nicht ausschließlich – zu Themen rund um root, z. B. auch diverse root-Guides
- Wikipedia (EN): [Rooting \(Android OS\)](#)
- Wikipedia (DE): [CyanogenMod](#) – einer der besten Gründe pro-root
- DroidWiki (DE): [Root](#) (und weitere interessante Themen rund um Android)
- App: [ROM Toolbox](#) – Werkzeugkasten für Wurzelmenschen (siehe Abbildung rechts), mit ROM-Manager, App-Manager, CPU-Manager, Font-Installer, Root-Browser (FileManager)...



Dateisysteme und Datenstrukturen

Dieses Kapitel wird ausführlich in der "Franzis-Edition" behandelt. Dennoch einige Details bereits an dieser Stelle. Zum Einen, damit der Leser weiß, was ihn dort erwartet – und zum Anderen, um zumindest ansatzweise die Grundlagen zum Verständnis einiger technischer Eigenheiten zu vermitteln.

Dateisysteme

Unter Android kommen verschiedene Dateisysteme zum Einsatz, die alle ihre Eigenheiten haben. Warum man sich nicht einfach auf ein Dateisystem festgelegt, und dieses dann durchgängig verwendet hat? Würde das nicht Vieles vereinfachen? Vielleicht. Aber von unseren Hosentaschen-Computern erwarten wir schließlich, dass sie sowohl performant laufen – als auch, dass der Akku möglichst lange durchhält. Zwei Dinge, die sich nicht gleichzeitig maximieren lassen – wie auch das Kapitel [Tuning](#) noch zeigen wird. Jedes einzelne Dateisystem hat seine Vor- und Nachteile: Was für das eine Einsatzgebiet gut geeignet ist, passt nicht unbedingt auch für alle anderen. Also wird mit Watson kombiniert.

Welche Dateisysteme finden sich nun auf dem Android-Gerät? Kommt ganz darauf an. Denn das entscheidet sich wieder einmal von Gerät zu Gerät, Android-Version zu Android-Version, Hersteller, ROM-Küche, und weiteren Eigenheiten. Doch jedes Android-Gerät verwendet mehrere (nicht jedoch unbedingt alle) der folgenden Dateisysteme:

EXT

Das [Extended Filesystem](#) wurde 1992 speziell für Linux entworfen, und seitdem ständig weiterentwickelt. Im Einsatz sind heute noch die Versionen 2-4 (ext2, ext3, ext4) – die allesamt auch unter Android anzutreffen sind. Mit Version 3 wurde das Journal eingeführt (was das System sicherer und die Überprüfung schneller macht). Ext4 legte dann in Sachen Performance noch einmal zu.

procfs

Bei [procfs](#) handelt es sich um ein virtuelles Dateisystem, das zur Ausgabe und Änderung von System- und Prozessinformationen dient (daher auch der Name: vom englischen *process*).

tmpfs

Der Name lässt es bereits vermuten: Bei [tmpfs](#) handelt es sich um ein temporäres Dateisystem. Zum Einsatz kommt es auf RAM-Disks, also im Arbeitsspeicher – für Daten, die einen Reboot nicht überleben müssen/sollen.

reiserfs

Das [Reiser File System](#) ist unter Android eher selten zu finden (obwohl es von einigen [Custom-ROMs](#) eingesetzt wird). Es war das erste Journaling-Dateisystem, welches vom Linux-Kernel direkt unterstützt wurde. Im Alltag trifft man überwiegend nur noch die neueren Versionen 3 und 4 an.

vfat

VFAT steht für **V**irtual **F**ile **A**llocation **T**able, und kann sowohl auf FAT16 als auch FAT32 aufsetzen. Unter Android wird es ziemlich ausschließlich für SD- und eMMC- Karten eingesetzt.

YAFFS2

YAFFS ist wieder einmal eine Abkürzung aus dem englisch-sprachigen Bereich. Wie so viele dieser mit YA beginnende Akronyme (Linuxer kennen das bereits), haben wir es auch hier mit "einfach etwas anderem" zu tun: **Y**et **A**nother **F**lash **F**ile **S**ystem. Es wurde speziell für den Einsatz mit **NAND-Flash** entwickelt – eben jenem Speichertyp, der in fast allen mobilen Geräten (also auch in unseren Androiden) verbaut ist. Daher kennt **YAFFS** die entsprechenden Eigenheiten, und kann so für eine längere Lebensdauer sorgen. In allen halbwegs aktuellen Android-Geräten wird die zweite Version davon eingesetzt – also **YAFFS2**.

Auf die speziellen Eigenheiten der einzelnen Dateisysteme, insbesondere in Bezug auf Android, geht – wie bereits kundgetan – die "Franzis-Edition" dieses Buches näher ein.

Datenstrukturen

Während wir uns unter [Dateisysteme](#) dem Speicher eher von der Seite der dahinter stehenden Technologie und Fragen wie *Warum ist YAFFS2 so besonders gut für den Flash-Speicher geeignet?* genähert haben, geht es jetzt vielmehr um das "Wo": Wo ist welches Medium eingebunden? Wo finde ich welche Daten? Auch dafür gilt wieder: Grobes zum Verständnis hier – ausführlicheres, vertiefendes, und hintergründigeres in der "Franzis-Edition" des Buches.

Eingebundene Dateisysteme

Der "einfache Anwender" weiß einfach: Da ist Speicher drin, und darauf liegen sicher die ganzen Dateien. Der Anwender mit ein wenig mehr technischem Hintergrund kann sich zumindest denken, dass der interne Speicher wohl auf einem anderen "Laufwerk" liegt als die SD-Karte. Aber wer würde schon vermuten, dass es insgesamt 15 oder mehr so genannte "Mount Points" gibt – wobei sich hinter jedem ein eigenes Dateisystem verbirgt? Kein Witz, das ist tatsächlich der Fall! Einige davon seien hier kurz aufgeführt – wobei dies nur exemplarisch ist, und nicht auf jedem Gerät etc. übereinstimmen muss:

Mount-Point	Typ	Kommentar
/cache	ext3	Anwendungs-Cache
/data	ext3	Dalvik-Cache, Logs, Konfigurations-Dateien...
/data/ data	YAFFS2	Anwendungs-Daten
/dev	tmpfs	"Geräte-Dateien" (devices)
/mnt/ asec	tmpfs	"entschlüsselte" Version der mit <i>App2SD</i> auf die Karte verschobenen .apk Dateien

Mount-Point	Typ	Kommentar
/mnt/emmc	vfat	interne "SD-Karte" (eMMC)
/mnt/sdcard	vfat	externe SD-Karte
/proc	procfs	System- und Prozess-Informationen
/system	YAFFS2	vorinstallierte System-Dateien

Nicht bei allen Geräten/ROMs wird /data/data als separate Partition eingebunden – dann sind die hier abgelegten Daten natürlich auf der /data Partition angesiedelt. Die in obiger Tabelle bezeichnete Partition bzw. das Verzeichnis /mnt/emmc ist gar auf vielen Geräten gar nicht vorhanden: Mein HTC Wildfire etwa verfügt gar nicht über diesen zusätzlichen Speicher. Und unter /mnt/sdcard kann natürlich auch nur dann etwas zu liegen kommen, wenn auch eine SD-Karte eingelegt ist (es soll ja Geräte geben, die diese Möglichkeit gar nicht bieten). Auf allen Android-Geräten an genau diesen Stellen vorhanden sein sollten jedoch die standardisierten Mount-Points: /data, /dev, /proc, und /system.

Dass /dev und /proc eine besondere Rolle spielen, ist zumindest eingefleischten Linux-Anwendern bekannt: An ersterer Stelle kann man nachschauen, welche "Geräte" denn so erkannt wurden (so findet sich für die meisten anderen genannten Dateisysteme unter /dev/block das entsprechende "Block-Gerät", über das auf den physischen Speicher zugegriffen werden kann). Und unter Letzterem lassen sich so einige System-Informationen auslesen (unter /proc/mtd etwa finden sich Informationen zu den im NAND-Flash angelegten Partitionen, und unter /proc/sys/kernel kann man verschiedene Kernel-Parameter auslesen oder auch anpassen). Doch auch /system hat eine Besonderheit: Es wird im reinen Lese-Zugriff eingebunden – Änderungen sind hier also nicht (ohne weiteres bzw. ohne root-Rechte) möglich. In dieser Partition finden sich nämlich die System-Programme – aber auch all die tollen "Zwangs-Beglückungen", die uns Geräte-Hersteller und ggf. Netzanbieter so gern "zur Verfügung stellen".

Einen ganz speziellen Fall stellt auch das temporäre Dateisystem dar, welches unter /mnt/asec eingebunden ist. Hat man nämlich mit App2SD Apps auf die SD-Karte ausgelagert, werden diese verschlüsselt im Verzeichnis .android_secure/ gespeichert. In diesem verschlüsselten Zustand kann das System sie leider nicht ausführen. Daher werden sie zuvor entschlüsselt unter /mnt/asec abgelegt – wo sie bei einem Systemstart automatisch wieder verschwinden.

Weitere Partitionen

Es gibt noch einige weitere Partitionen, von denen u. a. die folgenden noch interessant sein dürften:

Partition	Kommentar
/boot	Kernel & RAM-Disk
/recovery	Alternative Boot-Partition für Recovery Mode
/sd-ext	kein Standard; bei App2SD+ als "ausgelagerte /data Partition" genutzt

Einige wichtige Verzeichnisse

Während etwa an allem, was sich unterhalb von /system befindet, nur wirklich "fachkundiges Personal" sich vergreifen sollte – finden sich so einige interessante Verzeichnisse unterhalb von /data. Etliche davon werden für verschiedene Protokoll-Funktionen (auch als "Logging" bekannt) genutzt; diese sind unter [Andere Logdateien](#) aufgeführt.

Unter /data/data befinden sich dann die von den Apps verwalteten Daten – wobei jeder App ein eigenes Verzeichnis zugewiesen ist, auf das (normalerweise) nur sie selbst Zugriff hat (dass manch eine App diesen Schutz für die eigenen Daten aushebelt, wurde ja bereits zum Thema Sicherheit unter [App-Daten](#) gezeigt):

```

/data/data/com.example.demoapp
├── cache
│   ├── webviewCache
│   │   ├── 027e59a0
│   │   └── 057606c4
│   └── Verzeichnis
│       ├── Verzeichnis
│       ├── Cache-Datei
│       └── Cache-Datei
├── databases
│   └── example.db
│       └── SQLite-Datenbank
├── lib
│   └── Verzeichnis
├── shared_prefs
│   └── Verzeichnis
└── example.xml
    └── Konfigurations-Datei

```

So in etwa könnte das für eine "Beispiel-App" aussehen – es können aber durchaus weitere Verzeichnisse hier vorhanden sein, oder nicht einmal alle aufgeführten. Unsere Beispiel-App hat also...

- ...ein Cache-Verzeichnis, und darin noch ein paar Dateien von Seiten, die aus dem Web geladen wurden
- ...ein Datenbank-Verzeichnis mit einer SQLite-Datenbank (es können natürlich auch mehrere sein), in der sie strukturiert Daten in mehreren Tabellen verwalten kann
- ...ein ominöses Bibliotheks-Verzeichnis ohne Inhalt
- ...ein Verzeichnis mit Konfigurations-Dateien im XML-Format

Die Zugriffs-Berechtigungen sollten hier normalerweise etwa wie folgt aussehen:
 -rw-r----- für Dateien (der Eigentümer – also die App selbst – darf also lesen und schreiben, die Gruppe noch lesen, und der Rest hat hier nichts zu suchen),
 sowie maximal drwxr-x--x für Verzeichnisse (Eigentümer darf lesen, schreiben und ausführen – für Verzeichnisse bedeutet das: Hineinwechseln – die Gruppe noch lesen und ausführen, der Rest allenfalls noch ausführen); wobei "der Rest" hier eigentlich gar nichts mehr verloren hat.

Einige ausgewählte weitere Verzeichnisse (Quelle: Mein Motorola Milestone 2, als es noch Stock-Firmware mit Android 2.2.2 hatte) möchte ich hier kurz aufführen – für den Rest sowie weitere Details hingegen wieder auf die "Franzis-Edition" verweisen:

```

/data
├── backup
│   ├── com.android.internal.backup.LocalTransport
│   ├── com.google.android.backup.BackupTransportService
│   └── pending
├── dalvik-cache
│   └── (.dex (Dalvik EXecutable) Dateien)
└── local
    └── Zwischenspeicher für zu
        installierende .apk Dateien

```

location	Der Location-Cache ("Ich weiß, wo Du...warst!")
misc	Verschiedene Konfigurations-Dateien
bluetooth	
bluetoothd	
dhcp	
keystore	
systemkeys	
vpn	
wifi	
preinstall_md5	MD5-Hash Dateien einiger vorinstallierter Apps
system	Konfigurations- und Log-Dateien
dropbox	Log-Dateien von Dropbox-Syncs
registered_services	XML-Dateien mit Informationen zu verfügbaren (registrierten) Services
security	Sicherheits-Zertifikate (binär)
shared_prefs	Konfigurations-Daten (u. a. zu gesammelten Log-Dateien)
sync	Einstellungen von "Kontakte & Synchronisation": Welche Dienste sollen...?
usagestats	Nutzungs-Statistiken

Eine Überraschung war für mich, dass das Verzeichnis `/data/system/dropbox` gut mit Protokollen (Zeitstempel für Beginn/Ende der Synchronisation) gefüllt war, obwohl ich nicht einmal ein Dropbox-Konto habe! In den `registered_services` war auch kein Dropbox aufgeführt...

Im Verzeichnis `/data/system` finden sich überdies Dateien wie `accounts.db` (die Account-Datenbank mit konfigurierten Accounts, Authentifizierungs-Tokens, und mehr), `appwidgets.xml` (Launcher und konfigurierte Widgets), `batterystats.bin` (Akku-Statistiken), `packages.list` / `packages.xml` (installierte Apps / App Permissions), und weitere.

Externer Speicher

Auf der SD-Karte sieht es auch immer recht wild aus – und so manch einer mag sich nicht nur einmal gefragt haben, ob hinter dem ganzen Wildwuchs eine definierte Verzeichnis-Struktur verborgen sein könnte. Die Antwort lautet: Teilweise. Einige Standards sind etwa in der [Entwickler-Dokumentation](#) festgelegt, andere lassen sich durch Vergleichen verschiedener Geräte "erraten".

Verzeichnis	Kommentar
Android/data/ <package_name>/	entspricht <code>/data/data/<package_name></code> . Aktuelle Android-Versionen löschen dieses Verzeichnis bei Deinstallation der zugehörigen App.
Music/	Was der Medien-Scanner hier findet, stuft er als Musik ein
Podcasts/	... als Podcast ein
Ringtones/	... als Klingelton
Alarms/	... Alarmton
Notifications/	... Benachrichtigungston
Pictures/	Bilder (außer die der integrierten Kamera)
Movies/	Filme (außer die des integrierten Camcorders)
Download/	verschiedene Downloads
dcim/	Fotos/Videos der integrierten Kamera

Nicht immer wird sich an diese Strukturen gehalten – doch geben sie zumindest eine grobe Idee. Dummerweise machen die hier benannten Verzeichnisse meist weit weniger als die Hälfte der vorhandenen aus...

System-Info

Der Google Playstore bietet zahlreiche Tools, welche die verschiedensten Information zum System zu Tage fördern. Manche dieser Informationen möchte man gern ständig im Blick haben, andere benötigt man seltener. Zur ersten Kategorie gehören vielleicht Dinge wie CPU-Auslastung oder verfügbarer Speicher: Da bieten sich zweifellos Widgets an. Diese eignen sich allerdings weniger für ausführlichere Informationen (wie etwa eine Liste laufender Prozesse).

Es ist nicht einfach, alles eindeutig zu gruppieren – dazu gibt es zu viele Überschneidungen. Ich hoffe dennoch, dass mir eine passende Zusammenstellung gelungen ist.

Systeminfo-Widgets

Und schon beim ersten Tool scheitert es, da es nahezu alle Bereiche umfasst. Aber es ist sauber in einzelne Pakete aufgegliedert. Die Rede ist von *Elixier 2*, dessen Basis-Paket im Kapitel [Ausführliche System-Infos](#) vorgestellt werden soll. Sehr vorbildlich hat der Entwickler den Zugriff auf persönliche Informationen in ein [Personal Add-On](#) ausgegliedert; so sind die geforderten Permissions der [Elixier 2 Widgets](#) (rechtes Bild) zwar entsprechend dem, was sie im Auge behalten sollen, nicht wenige – dafür aber dennoch recht unbedenklich.

Der in diesem Kapitel interessante Teil hat im rechten Screenshot einen blauen Hintergrund, und zeigt wichtige Informationen zum System auf einen Blick. Dabei hat man die Auswahl aus einigen Datenquellen: Akku (Füllstand, Spannung, Temperatur), externer/interner Speicher, CPU (Auslastung und/oder Frequenz), RAM, Anzahl laufender Prozesse, Uptime, Signalstärke (Wifi, Mobilfunk), Traffic (Wifi, Mobilfunk) – aber auch (roter Bereich) Anzahl ungelesener Mails/Kurznachrichten, verpasste Anrufe und mehr. Und um dem nächsten Kapitel gleich vorzugreifen: Auch eine ganze Reihe von "Toggle-Widgets" (also "Schnellumschalter", siehe obere Bildhälfte) sind mit an Bord, wie etwa APN, Wifi, Wifi Hotspot (aka [Tethering](#)), Bluetooth, GPS, Flugzeugmodus, und sogar SD-Mount sowie SD-Refresh (Media-Scan anstoßen) stehen hier – unter anderem – zur Auswahl. Da bleibt fast kein Auge trocken. Und bei den Widgets stört es auch wenig, dass es wohl keine Unterstützung für die deutsche Sprache gibt: Ausgewiesen sind nämlich nur Englisch und Ungarisch...

Ein unheimlich erfolgreiches Päckchen, dieses *Elixier 2*. Das best bewerteste Päckchen in diesem Bereich, nebenbei bemerkt. Und ich traue mich kaum zu schreiben, was denn wohl das zweitbest bewertete Päckchen sein könnte – das trägt nämlich den Namen [Elixier](#), und ist die Vorgänger-Version (bei der die Widgets noch nicht in einer separaten APK-Datei ausgeliefert wurden)...





Auch das an dritter Stelle folgende [Mini Info](#) (linkes Bild) fällt in diese Misch-Kategorie: Für den Home-Screen bietet es einige Widgets mit den wichtigsten Informationen, wie Akku (Reststand, verbraucht, Spannung, Temperatur, Status), RAM/CPU (in Nutzung bzw. verfügbar) und Speicherplatz (intern/extern; belegt, frei, total). Die dahinterstehende App stellt dann bei Bedarf weitere Informationen zur Verfügung – sowie eine Bar mit "Toggle-Widgets". Die Info-Widgets lassen sich, wie auf dem Screenshot zu sehen, nach Gusto gruppieren. Ebenso ist das Festlegen eines Hintergrundes möglich: Von völlig transparent, über halbtransparent/Glas bis hin zur "Vollfarbe".

Zum Ausprobieren gibt es *Mini Info* gratis, der volle Funktionsumfang lässt sich per Donation (ca. anderthalb Euro) freischalten. Übrigens gibt es für diese App einen super Support mit kurzen Reaktionszeiten.

Auch wenn sich diese Apps wieder nicht ausschließlich dem Thema "Systeminfo-Widgets" widmen, finde ich die Kombination sehr sinnvoll: Uhr und Wetter (im Stil von *HTC Sense*, daher auch der Name), und unmittelbar darunter die wichtigsten System-Infos eingeblendet – so kommt [Sense Analog Clock Widget 4x2](#) (rechtes Bild; auch in anderen Varianten mit 24-Stunden-Display, oder im Glass-Look erhältlich – einfach einmal die weiteren Apps dieses Entwicklers durchforsten) daher. Ja, auch die Wettervorschau für die nächsten Tage ist integriert – doch da es hier ja um die Systeminfo-Widgets gehen soll: Das Wichtigste wird dargestellt (RAM, interner/externer Speicher, Temperatur, und Akku-Stand), detailliertere Infos gibt es beim Antippen. Auch lässt sich konfigurieren, welche App beim Antippen der Stunden- oder der Minutenzahl geöffnet werden soll (Kalender und Wecker bieten sich hier an), sodass kein Platz "verschwendet" wird...

Eine Auswahl weiterer Widget-Sammlungen finden sich in [dieser Übersicht](#) bei AndroidPIT.



Schnellumschalter

Hier könnte ich es mir jetzt einfach machen und schreiben: "Siehe [oben](#)" – zumindest die vorderen Plätze sind sogar identisch (*Elixier 2* und *Elixier*). Ebenso ist oben genanntes *Mini Info* recht weit oben mit dabei. Aber einige Kandidaten haben sich speziell auf das "schnelle Umschalten" spezialisiert, und sollen daher auch in diesem Kapitel genannt werden.

So nimmt etwa [MySettings](#) (Bild rechts) vergleichsweise viel Platz ein – ist aber nach den beiden *Elixieren* in dieser Kategorie die am besten bewertete App. Wer genau hinschaut, erkennt hier sogar noch grundlegende System-Infos, die am oberen Rand eingebettet werden. Mit dieser App lässt sich nicht nur schnell umschalten, auch Helligkeit und diverse Lautstärke-Regler lassen sich mit ihr schnell justieren. Somit ist es eigentlich eine Kreuzung aus "Schnellumschalter" und "Schnellkonfigurator". Wie wir nun bereits mehrfach gesehen haben, sind die Grenzen zwischen den einzelnen Bereichen recht fließend...



Eine feine Sache ist auch [Dazzle](#) (linkes Bild). Hier lassen sich beliebige Kombinationen aus "Toggles" in einem Widget kombinieren – auch sieben Schalter in einer Leiste (4x1 Widget) sind kein Problem in der Darstellung (anders aussehen mag das in Bezug auf die Treffsicherheit von Wurstfingern). Was auf dem Screenshot auch sehr schön zu sehen ist: Man kann durchaus mehrere Widgets anfertigen, die unabhängig voneinander konfigurierbar sind (die Dazzle-Widgets sind auf dem Screenshot ausschließlich die mit dem schwarzen Hintergrund). Das ergibt größtmögliche Flexibilität.

Darüber hinaus geht hier mehr als bloßes "An" und "Aus": Auf Wunsch gibt es auch "Pop-Up-Schieberegler" etwa für stufenlose Regelung der Bildschirm-Helligkeit oder Klingelton-Lautstärke. Wobei für den letzteren auch ein Multi-Schalter möglich ist: An/Aus/Vibrieren, plus Knopf für Benutzerwert. Wer mag, kann auch einfach den Mobilfunk abschalten – etwa um ungestört von Anrufen und eingehenden Kurznachrichten im WLAN zu surfen...

Eine der beliebtesten Lösungen in diesem Umfeld ist sicher nicht ohne Grund [Widgetroid 2](#), welches einen schier aus den Latschen hauen kann. Nicht nur lässt sich hier alles bis ins Feinste anpassen (von Hintergrundfarbe und Transparenz über benutzerdefinierte Farben und Label, unterschiedliche Widget-Größen (bis zu 10 Schaltflächen auf einem Widget) bis hin zu eigenen Hintergrundbildern für die Widgets), und die Widget-Auswahl neben den "üblichen Verdächtigen" auch Dinge enthält wie das Neueinlesen der SD-Karte, Reboot (bei gerooteten Geräten gar mit Auswahl aus Recovery, Bootloader, Fastboot, Normal, und Herunterfahren), Tethering (WLAN/USB), beliebige App-Verknüpfungen oder gar Kontakte (optional mit Popup-Auswahl aus Anrufen, SMS, E-Mail). Nein: Man kann sich auf den Widgets auch Dinge wie Akkustand und -temperatur, Speicher oder RAM anzeigen lassen.

Die Krönung ist jedoch: Das Ganze lässt sich auch in die [Notification Area](#) einbauen (siehe rechtes Bild)! Auf diese Weise hat man seine Schnellumschalter jederzeit durch einfaches Herunterziehen der Statusleiste im Zugriff – egal, auf welchem Screen oder gar in welcher App man sich gerade befindet. So dürfte *Widgetroid 2* die am umfangreichsten anpassbare App in diesem Bereich sein.

Damit könnte nun die Vorstellung der Schnellumschalter wirklich enden. Und das tut sie auch – mit dem Hinweis, dass sich bei Bedarf weitere Apps in [dieser Übersicht](#) finden...



Ausführlichere System-Infos

Auch an dieser Stelle geht der Verweis zuerst "nach [oben](#)": Wie schon bei den Widgets, steht hier ein *Elixir* an erster Stelle. Und zwar die Basis-App [Elixir 2](#), die – wie rechts im Bild zu sehen – Informationen zu vielen Bereichen zur Verfügung stellen möchte. Dazu zählen u. a. Geräte-Informationen wie Akku (Ladezustand, Spannung/Strom, Temperatur), interner und externer Speicher (insgesamt/frei, wo eingebunden), CPU (Auslastung, Frequenz), RAM (insgesamt/frei), Telefonie (Netzwerk-/Telefonmodus, Provider, APN); installierte Anwendungen (wo installiert, wie viel Platz belegt die App, wie viel ihre Daten, welche Version); welche Prozesse/Services laufen (und verwenden wie viel RAM/CPU etc.) – aber auch Daten der Sensoren. Ebenso lassen sich diverse Systemeinstellungen (wie etwa Bildschirmhelligkeit oder Lautstärke) anpassen, oder Aktionen wie Cache-Bereinigung, SD-Karte ein-/ausbinden oder Bluetooth-Scan durchführen. Als würde das noch nicht reichen, greift die App auch noch in das Kapitel [Logcat](#), und zeigt die Systemlogs an. Großer Rundumschlag – oder Schweizer Offiziersmesser. Und dabei habe ich noch lange nicht alles aufgezählt...



Recht umfangreiche Informationen zum System stellt auch [Android System Info](#) (linkes Bild) bereit. Diese bestehen u. a. aus einem "Dashboard" mit den wichtigsten Eckdaten, einem Task-Manager, der sowohl Task-Switcher als auch Task-Killer beinhaltet, einer Anwendungs-Verwaltung für die installierten Apps, einem farblich aufbereiteten System-Log (noch jemand, der dem Kapitel [Logcat](#) vorgreift), sowie weiteren Informationen rund um das System. Etliche Nutzer berichten, was ihnen besonders gefällt: Zahlreiche Informationen, und doch übersichtlich aufbereitet. Entsprechend weit oben ist es auch auf der Bewertungsliste platziert: Durchschnittlich 4,6 Sterne bei über 20.000 Bewertungen. Trotzdem hinter *Elixir 2*, welches über 4,8 Sterne durchschnittlich erhielt.

Vom Aufbau her vergleichbar wäre **OS Monitor** – bringt allerdings noch ein paar Schmankerln mit. Wie etwa die auf dem rechten Screenshot zu sehende "grafische Whols-Abfrage": Wohin hat sich diese App gerade verbunden? Aha, da saßten die...

Im Hintergrund erkennt man bei genauerem Hinsehen noch die "Reiterlein". Hinter selbigen verbergen sich eine Prozessliste (die sich nach verschiedenen Kriterien wie beispielsweise CPU oder Speicherverbrauch sortieren lässt, und auch einen Task-Killer beinhaltet), eine Übersicht über verfügbare Netzwerk-Interfaces (mit, sofern verfügbar, Details zur **MAC-Adresse**, zugeteilter IP, sowie bereits übertragener Datenmenge), aktuelle Datenverbindungen (welche App wohin, mit welchem Netzwerk-Protokoll?), eine Zusammenfassung "verschiedener" Informationen (CPU, Akku, Speicher) sowie – ja, auch hier wieder – ein farblich aufbereitetes Systemlog.



Zusätzlich lässt sich die CPU-Auslastung mit einem kleinen Icon in der Statusleiste einblenden – entweder einmalig manuell, oder automatisch bei jedem Systemstart aktiviert. Sofern der Androide gerootet ist, lässt sich auch die Taktfrequenz der CPU anpassen. Bei 170kB APK-Datei (installiert gut 300kB) ist die App mit diesen Leistungsmerkmalen recht kompakt geraten. Die Permission-Liste ist ebenfalls sauber, und die Bewertungszahl (mit durchschnittlich 4,5 Sternen bei fast 5.000 Bewertungen) ist topp.

Weitere Apps zur Beschaffung von System-Informationen finden sich in **dieser Übersicht**. Ein paar grundlegende Informationen findet man übrigens auch mit Bordmitteln, indem man vom Home-Screen aus über *Menü* → *Einstellungen* geht und "Telefoninfo" auswählt...

System Logs

Spätestens wenn etwas nicht so tut, wie es sollte, kommt die Frage auf: "Was geht da eigentlich ab?" Neugierige, Entwickler, und auch technisch interessierte stellen diese Frage selbst dann, wenn alles reibungslos läuft.

Auskunft darüber geben die Log-Dateien des Systems – von denen es in der Tat einige gibt. Und ebenso gibt es eine ganze Reihe von Möglichkeiten, an deren Inhalte zu kommen. Einige davon sollen hier vorgestellt werden. Doch bevor es an die Details geht, möchte ich eine grobe Übersicht voranstellen: Auf welche Arten lässt sich auf diese System-Protokolle zugreifen?

Methode	Vorteile	Nachteile
Per App	Kein weiteres Gerät und keine spezielle Software auf dem Arbeitsrechner notwendig	kleiner Bildschirm (insbesondere auf Smartphones), eingeschränktes Multitasking

Methoden	Vorteile	Nachteile
Via ADB	Bequemes Arbeiten vom PC aus, einfaches Multitasking	PC mit installiertem SDK notwendig
Via SSH	Bequemes Arbeiten vom PC aus, einfaches Multitasking	?

Zugriff per App

Ab und an benötigt auch der einfache Anwender einen Auszug aus den System-Logs. Etwa wenn eine App sich "seltsam verhält" oder gar nachvollziehbar abstürzt, und man dies dem Entwickler mitteilen möchte: Der nämlich freut sich über derartige Details. Weiter oben habe ich bereits aufgezeigt, dass einige Apps diese Informationen sehr wohl anzeigen – nur wie bekommt man selbige dann zum Entwickler?

Die bekannteste Antwort darauf heißt sicher [aLogcat](#) (rechtes Bild), ist mit 4,6 Sternen bei über 1.500 Bewertungen recht beliebt – und sollte auf keinem Androiden fehlen. Diese App startet man, leert die Einträge kurz über den abgebildeten "Clear Log" Button, provoziert den Fehler in der fraglichen App, kehrt zu *aLogcat* zurück, drückt auf "Pause" (da wo jetzt "Play" steht) – und nun hat man die Wahl, den Log-Ausschnitt mit "Save Log" auf die SD-Karte zu befördern, oder über den "Send" Button per Mail direkt zum Entwickler zu befördern. Das schaffen auch technisch weniger versierte!



Alternativ dazu wäre noch der [Log Collector](#) zu nennen, der ähnlich funktioniert. Oder auch der links abgebildete [Bug Reporter](#), der zusätzlich zum Auszug aus den Systemlogs auch noch Informationen zum verwendeten Android-System sammelt und in die Mail an den Entwickler einschließt – das erspart das mühsame manuelle Zusammensuchen und stellt sicher, dass man kein wesentliches diesbezügliches Detail vergessen hat.

Weitere mögliche Alternativen findet man, indem man erstgenanntes *aLogcat* bei [AndroidPIT](#) aufsucht, und in die Liste vergleichbarer Apps schaut. Da fast alle gleich arbeiten und aussehen, habe ich dazu keine separate Übersicht erstellt.

Zugriff via ADB oder SSH

Doch das ist bei Weitem nicht alles, was das System an Logs zu bieten hat – sondern lediglich das, was dem Anwender recht bequem zur Verfügung steht.

Wer gern noch etwas tiefer graben möchte, hat dazu durchaus die Möglichkeit: Entweder mit einer Terminal-App direkt auf dem Androiden – oder aber bequemer vom PC aus, was jedoch die Installation der Android Entwicklungs-Umgebung ([SDK](#)) – oder alternativ den Einsatz eines [SSH](#)-Servers auf dem Androiden – erforderlich macht. Bevor ich also auf die einzelnen Logs eingehe, kurz ein paar Worte dazu.

Bei auf dem Arbeitsrechner installiertem SDK, und mit aktiviertem USB-Debugging (dies lässt sich im Systemmenü des Androiden unter *Anwendungen* → *Entwicklung* ein- und ausschalten) per USB-Kabel an selbigem angeschlossenen Android-Gerät lässt sich über [ADB](#) auf letzteres zugreifen. Der Name "Android Debug Bridge" (sowie die Stelle, an der die Einstellung am Android-Gerät erfolgen muss) legen bereits nahe, dass dies hauptsächlich zur Entwicklung gedacht ist. Um Befehle vom Arbeitsrechner aus direkt auf dem Androiden ausführen zu können, stellt das SDK den Befehl `adb shell` zur Verfügung. Wer also folgendes gern bequem über die ADB ausprobieren möchte, setzt jedem Befehl einfach ein `adb shell` voran – etwa `adb shell logcat`. Die Ausgabe erfolgt dann natürlich im Terminal-Fenster des Arbeitsrechners.

dmesg

Was wird nun vom Android-System eigentlich alles protokolliert? So allerhand. Nur wird es nicht lange aufgehoben. Die meisten Protokolle liegen auf einem temporären Dateisystem (`tmpfs`), das bei Systemstart im RAM angelegt wird. Nach einem Neustart des Androiden sind also alle zuvor angelegten Einträge weg. Dafür werden aber neue geschrieben – und zwar bereits wenn das System hochfährt. Da Android auf einem Linux-Kernel läuft, kommt Linux-Anwendern natürlich sofort der Befehl `dmesg` in den Sinn – und tatsächlich findet sich das auch hier wieder. Da die Ausgaben hier recht umfangreich sind, empfiehlt sich die Umleitung in eine Datei, etwa wie folgt: `dmesg > /mnt/sdcard/dmesg.log`.

```
$ dmesg
<6>[82839.126586] PM: Syncing filesystems ... done.
<7>[82839.189056] PM: Preparing system for mem sleep
<4>[82839.189361] Freezing user space processes ... (elapsed 0.05 seconds) done.
<4>[82839.240661] Freezing remaining freezable tasks ... (elapsed 0.00 seconds) done.
<7>[82839.242279] PM: Entering mem sleep
<4>[82839.242889] Suspending console(s) (use no_console_suspend to debug)
<7>[82839.252410] vfp_pm_save_context: saving vfp state
<6>[82839.252716] PM: Resume timer in 26 secs (864747 ticks at 32768 ticks/sec.)
<6>[82842.091369] Successfully put all powerdomains to target state
<6>[82842.092468] wakeup wake lock: wifi_wake
<snip>
```

Wie die meisten anderen Protokoll-Dateien, so handelt es sich auch bei `dmesg` um einen "Ringpuffer" mit fester Größe: Ist er voll, und muss etwas neues geschrieben werden – dann fliegt etwas altes raus. `dmesg` ist ja keinesfalls ein "Boot-Log", sondern vielmehr die Log-Datei des Kernels – und der läuft ständig im Hintergrund. Je länger also der letzte Bootvorgang her ist, desto wahrscheinlicher sind auch die entsprechenden Einträge bereits durch neuere Ereignisse überschrieben worden.

logcat

Bereits im Zusammenhang mit einigen grafischen Tools wurde logcat genannt. An der Kommandozeile lässt sich diesem noch einiges und viel präziser entlocken. Eine ausführliche Beschreibung dieses Befehls ist auf der [Hilfe-Seite](#) der Android-Entwickler-Website zu finden – dennoch hier ein paar kurze Hinweise.

Ein kleines Problem zeigt sich nämlich gleich beim ersten Aufruf:

```
$ logcat
Unable to open log device '/dev/log/main': Permission denied
```

Anders als mit den oben genannten Apps, scheint der shell User also nicht so ohne weiteres Zugriff auf logcat zu erhalten – zumindest bei Zugriff über [SSH](#) benötigt man also root-Rechte. Dann geht es aber:

```
$ su
# logcat
----- beginning of /dev/log/system
<snip>
D/WifiWatchdogService( 3457): (android.server.ServerThread) IzzySoft (24:65:11:67:d7:6d) does not require the watchdog
I/ActivityManager( 3457): Starting: Intent { act=android.settings.WIFI_SETTINGS cat=[android.intent.category.DEFAULT] flg=0x10000000 cmp=com.android.settings/.wifi.WifiSettings } from pid 3529
W/InputManagerService( 3457): Starting input on non-focused client com.android.internal.view.IInputMethodClient$Stub$Proxy@409f6128 (uid=1000 pid=2760)
<snip>
```

Über den Parameter -b lässt sich der Puffer angeben, aus dem man Informationen erhalten möchte. So liefert beispielsweise ein logcat -b events Daten aus dem Ereignis-Protokoll, während logcat -b radio Details zum "Radio" (also dem Mobilfunk-Element) preisgibt. Auch diverse Einstellungen lassen sich mit diesem Befehl steuern – etwa die Größe des zu verwendenden Ringpuffers. Diese gelten natürlich nur bis zum nächsten Gerätestart.

```
$ su
# logcat -b events
I/am_create_service( 3457): [1085416560,nitro.phonestats/.widget.WidgetProvider4x1$WidgetUpdateService
I/am_destroy_service( 3457): [1085416560,nitro.phonestats/.widget.WidgetProvider4x1$WidgetUpdateService
I/notification_cancel( 3457): [nitro.phonestats,4,0]
I/db_sample( 4002): [/data/data/com.lbe.security.lite/databases/lbe_trafficmon.db,INSERT OR REPLACE INTO
uid_traffic(wifitx, wifirx, date_uid, ce,0,com.lbe.security.lite,1]
I/force_gc( 3721): bg
I/dvm_gc_info( 3721): [8099846449757718054,-9031259007116011474,-4008901822536468439,0]
<snip>
```

Jede Zeile der Logcat-Ausgabe beginnt mit einem Loglevel-Hinweis, gefolgt von einem Schrägstrich, gefolgt vom Ursprung des Eintrags – beispielsweise D/WifiWatchdogService(3457). Daran schließt sich sodann die eigentliche Meldung an. Folgende Log-Level gibt es: **V**erbose, **D**ebug, **I**nfo, **W**arning, **E**rror, **F**atal, **S**ilent. Der Wert in Klammern gibt die Prozess-ID an, sodass auch Querverweise zu anderen Protokollen möglich sind.

dumpsys und dumpstate

Diese beiden Befehle geben ausführliche Informationen zum System: Dienste, Speicher, Broadcasts, anstehende Intents, Activities, Prozesse, Account-Informationen; Informationen über das Gerät, Build, Radio, Netzwerk, Kernel-

Details... Auch hier empfiehlt sich, die Ausgaben in eine Datei schreiben zu lassen, da sie überaus umfangreich sind.

Für beide Befehle werden keine root-Rechte benötigt - jedoch fehlen bei Ausführung ohne selbige einige Details:

```
$ dumphsys
Currently running services:
  LocationProxyService
  SurfaceFlinger
  accessibility
  account
  activity
<snip>
DUMP OF SERVICE account:
Accounts: 1
  Account {name=xxxxxxx@googlemail.com, type=com.google}
<snip>
DUMP OF SERVICE alarm:
Permission Denial: can't dump AlarmManager from from pid=7274, uid=10038
<snip>
DUMP OF SERVICE batteryinfo:
Battery History:
  -10h32m25s939ms 099 23c30040 status=discharging health=good plug=none temp=280 volt=4117 +scre
en +wifi +wifi_running +wifi_full_lock +wifi_scan_lock +wake_lock +sensor signal_strength=great
  -10h32m19s441ms 099 03c10040 -screen -wake_lock
<snip>

$ dumpstate
=====
== dumpstate: 2012-08-18 23:39:53
=====

Build: Gingerbread GWK74 - CyanogenMilestone2
Bootloader: 0x0000
Radio: unknown
Network: E-Plus
Kernel: Linux version 2.6.32.9-g5db7937 (a17140@zch68lnxdroid63) (gcc version 4.4.0 (GCC) ) #1 PREEM
PT Fri Sep 16 17:57:00 CST 2011
Command line: (unknown)

----- MEMORY INFO (/proc/meminfo) -----
MemTotal:      487344 kB
MemFree:       10436 kB
Buffers:       14136 kB
Cached:        145460 kB
<snip>
```

Unschwer zu erkennen, dass sich hier allerhand über System, Konfiguration, Dienste, und vieles mehr in Erfahrung bringen lässt.

bugreport

Was mag sich wohl hinter einem Befehl mit diesem Namen verbergen? Große Preisfrage. Antwort: Alles obige für Tippfaule. Gibt man bugreport im Terminal (oder adb shell bugreport via ADB) ein, flattern die Zeilen nur so über den Bildschirm - hier sollte also auf jeden Fall die Ausgabe in eine Datei umgeleitet werden (bugreport > /mnt/sdcard/bugreport.txt). Das Ergebnis sind dann alle von logcat, dumphsys und dumpstate gesammelten Informationen in einem

schönen Bugreport für den Entwickler – also genau das, was auch die oben beschriebene App *Bug Reporter* erzeugt.

Andere Logdateien

Außer den bislang genannten Standard-Logs gibt es noch einige Verzeichnisse, in denen sich Log-Dateien finden lassen. Da diese durchaus von Gerät zu Gerät (und auch zwischen verschiedenen Android-Versionen) unterschiedlich sein können, sind nicht alle hier genannten Verzeichnisse auf jedem Gerät vorhanden – die mit "AOSP" gekennzeichneten sollten jedoch zum Standard gehören. Aber selbst wenn das Verzeichnis existiert, kann es durchaus auch leer sein...

- /data/anr: Einige **Trace-Logs** landen u. U. hier (bei **Application Not Responding**, auch als "Force-Close" oder "Schließen erzwingen" bekannt)
- /data/dontpanic (AOSP): Crash-Logs mit **Traces**. Im Falle einer auftretenden Kernel-Panik finden sich hier zwei Dateien: `apanic_threads` verzeichnet den zu diesem Zeitpunkt aktiven Prozess/Thread, `apanic_console` enthält zusätzliche Informationen wie Traces und Register
- /data/kernelpanics (AOSP): Log-Einträge bei sog. "Kernel-Panic"
- /data/panic/panic_daemon.config: Konfigurations-Datei für den Panic-Daemon. Wenn unterstützt, lässt sich hier festlegen, wohin Panic-Logs bei manueller Auslösung (über Geräte-spezifische Tastenkombinationen) geschrieben werden (oft `/sdcard/panic_data/`).
- /data/panicreports: Weitere Panik-Protokolle werden bei einem "Not-Halt" hier abgelegt (AOSP).
- /data/tombstones: Lustig sind sie ja, die AOSP Entwickler: Auf das Grab (englisch "tomb") kommt ein Stein (englisch "stone"), also ein Grabstein ("tombstone"), damit man sich an den Toten erinnert. Stirbt also ein Prozess unerwartet (auch als "Absturz" oder "Crash" bekannt), kann er sich auch so einen "Grabstein bestellen" – der dann Informationen zu seinem Ableben enthält. Auf Linux/Unix Systemen nennt man dies "Core Dumps"...

Monitoring

Fast 5.000 Nutzer bewerteten es mit durchschnittlich 4,7 Sternen, es kostet gut zwei Euro: **SystemPanel** scheint mir hier die App der Wahl, um einen "Bösewicht" aufzuspüren. Beispielsweise wenn man wissen möchte, wer einem da gerade den Akku leer genuckelt hat: Vor vier Stunden war er noch randvoll, und jetzt ist die Hälfte verschwunden? **SystemPanel** hat da sicher die passenden Details zur Hand, wie im rechten Bild ersichtlich. Die Auswertung von oben nach unten: Zwischen 18 und 20 Uhr hat Google Maps die CPU mit ca. 5% Beschlag belegt – davor und danach gar nicht ("Process CPU Activity"). Eine externe Stromversorgung (Ladekabel) war in diesem Zeitraum nicht angeschlossen, sondern erst wieder ab ca. 21 Uhr ("Charging"). Ebenfalls im Zeitraum von 18 bis 20 Uhr verlor der Akku etwa die Hälfte seiner Ladung ("Battery Charge"), was sich so ziemlich mit der Aktivität von Maps deckt. Das Ganze scheint außerdem größtenteils "im Hintergrund" geschehen zu sein, denn laut "Device Usage" war der Anwender selbst im gleichen Zeitraum weniger aktiv. Da hätten wir also den Verursacher so ziemlich genau identifiziert, oder? Doch halt: Die gesamte CPU Auslastung ("CPU Activity") lag im fraglichen Zeitraum bei über 10%, also muss wohl noch jemand beteiligt sein...



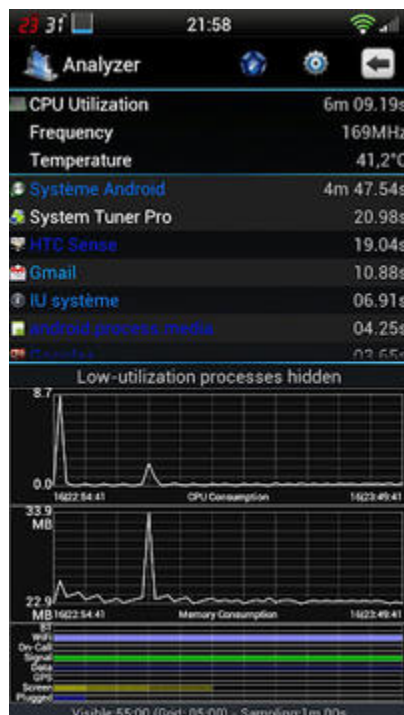
Eine derartige Auswertung ist nur in der Vollversion möglich, die das "Protokollieren" erlaubt (also im Hintergrund die Daten bis zu einer Woche lang festhält). Neben diesem System-Monitor bietet **SystemPanel** noch Geräte-Information, einen Task-Manager sowie einen App-(Un-)Installer. Einen ersten Eindruck kann man sich natürlich mit der Lite-Version verschaffen.



Doch auch mit der Lite-Version lässt sich bereits einiges anstellen. Auf der Suche nach Ressourcen-Fressern fällt gleich in der App-Liste der kleine senkrechte Balken links der App-Namen auf (siehe linkes Bild): Je höher dieser gefüllt ist, desto mehr CPU-Zyklen hat dieser Prozess während seiner Laufzeit beansprucht. Ist der Balken besonders hoch, und dies durch Art und Einsatz der App nicht gerechtfertigt? Dann wird es vielleicht Zeit, sich nach einer Alternative umzusehen.

Der nächste Blick gilt der Tatsache, welche App die CPU besonders lange mit Beschlag belegt hat. Dazu gibt die Übersicht der Top-Apps nach CPU (rechtes Bild) Auskunft. Eine App, die man nur gelegentlich verwendet, und die (eigentlich) in der restlichen Zeit auch keine Hintergrund-Aufgaben durchzuführen hat, sollte hier nicht auftauchen. Ansonsten gilt gerade gesagtes: Ist man als Anwender nicht selbst daran Schuld (es könnte ja auch eine Fehlkonfiguration vorliegen, oder die ungewünschte Hintergrund-Funktion lässt sich etwa in der App selbst deaktivieren): Der Playstore bietet in der Regel genügend ähnliche Apps an – sofern der Entwickler der App nicht Abhilfe schaffen kann.

Wonach könnte man noch schauen? Etwa nach dem Speicherverbrauch. Verbraucht eine App viel Speicher, heißt dies natürlich nicht automatisch, dass da etwas schief liegt: Die Sache kann durchaus einen guten Grund haben. Doch der von einer App belegte Speicher steht anderen Apps nicht mehr zur Verfügung – und wird der Speicher knapp, bremst dies das System aus. Die Relationen sollten also schon stimmen: Verbraucht etwa eine einfache Stoppuhr mehr Speicher als eine Office-Suite, passt da definitiv etwas nicht zusammen.



Ähnliches wie das nun ausführlich behandelte *SystemPanel* möchte auch *System Tuner* (linkes Bild) auf die Beine stellen. Auch diese App kann Aktivitäten im Hintergrund aufzeichnen und für eine spätere Analyse zur Verfügung stellen, RAM und Speicherdetails anzeigen, hat einen Task-Manager an Bord und sogar einen Terminal-Emulator, sodass man Befehle direkt an der Kommandozeile absetzen kann. Letzteres klingt ein wenig "nerdy"? Auf gerooteten Geräten lässt sich mit *System Tuner* auch der Cache der SD-Karte anpassen, der *OOM-Killer* konfigurieren, die CPU tweakten – man kann gar System-Apps in den "User-Space" verschieben oder gleich ganz deinstallieren. Das ist jetzt "nerdy"...

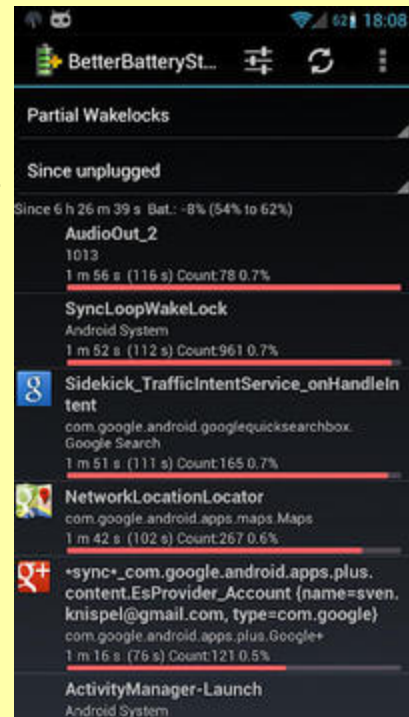
Mit dabei ist hier auch ein Widget, welches die wesentlichsten Eckdaten (CPU, RAM, Speicher) anzeigen kann. Auch einige Aktivitäten (Monitoring starten/stoppen, Task-Manager/Analyzer/Einstellungen öffnen) lassen sich von diesem aus steuern.

Weitere interessante Daten stellt auch [BetterBatteryStats](#) (rechtes Bild) bereit. So bringt diese App etwa die *WakeLock* Statistiken zurück, die bis einschließlich Froyo zum Bordgepäck gehörten. Der Name suggeriert es bereits: Hier wird jemand in den Wachzustand gesperrt – und zwar die CPU. Fordert also eine App ein *WakeLock* an (und erhält es zugeteilt), kann sich die CPU nicht schlafen legen, solange dieser aufrecht erhalten wird. So etwas schlägt sich natürlich nicht zuletzt im Akku-Verbrauch nieder.

BetterBatteryStats verrät uns also, welche Apps sich hier (aber auch in weiteren Dingen, die zum Akkuverbrauch beitragen) schuldig bekennen müssen. Dazu muss *BetterBatteryStats* selbst natürlich im Hintergrund mitlaufen – denn wie sonst soll die App an eben diese Daten kommen?

Auf zwei Dinge sollte hier besonders geschaut werden: Gleich auf der Startseite der App findet sich der Abschnitt *Deepsleep* ("Tiefschlaf"). Wer hat hier besonders viel CPU verbraucht? Der ist ein Kandidat zum Löschen/Ersetzen, [Deaktivieren des automatischen Startens](#), bzw. Abstellen/Herunterregeln der automatischen Synchronisation. Schwieriger wird es bei Einträgen wie *Moderate Signal* (kein optimaler Netzempfang), *Wifi On* oder *Wifi Running* (WLAN): Das lässt sich nicht eben mit einer anderen App ersetzen. Lösungen gibt es dafür trotzdem, wie unter [Automatisch \(De\)Aktivieren](#) beschrieben ist.

Ein weiterer Punkt sind die bereits erwähnten *Partial Wakes*, wie im rechten Bild zu sehen. Hier verbergen sich die Bösewichter, die der CPU schlaflose Tage und Nächte bereiten. Interessanter Kandidat im Screenshot: *NetworkLocationLocator*. Das ist der Dienst von *Google Maps*, der überwacht, wo sich der Androide (und somit i. d. R. sein Besitzer) wann aufhält – und diese Informationen im [Location Cache](#) ablegt, um sie später an Google zu übertragen. Wie man sieht, ist der an kurzer Akku-Laufzeit nicht gerade unschuldig.



Da wir gerade beim Monitoring sind: Interessant wären sicher auch Informationen darüber, welche App wann im Internet wohin unterwegs war. Diese Funktionalität bietet unter anderem die App *DroidWall*, auf die ich bereits zuvor beim Thema [Firewalls](#) näher eingegangen bin. Für Bluetooth-Aktivitäten trifft das in gleicher Weise auf *Bluetooth Firewall* im selben Kapitel zu. Weitere Apps zum System-Monitoring finden sich u. a. in [dieser Übersicht](#).

Konfiguration

*Über sieben Brücken musst Du geh'n,
Und an mehr als sieben Schraubchen dreh'n...*

Das Problem dabei ist nur häufig: Welche Schraubchen, und wo sind sie zu finden? Einige davon möchte ich im Folgenden kurz benennen:

Konfiguration mit Bordmitteln

Für die grundlegenden Dinge muss man nicht gleich auf zusätzliche Apps zurückgreifen. Ein wenig "Graben" mit den Bordmitteln fördert so manches zu Tage, dessen Existenz man nicht einmal geahnt hätte. Eine gute Empfehlung bei Geräte- bzw. ROM-Wechsel ist daher: Erst einmal ganz systematisch das gesamte Einstellungs-Menü durchgehen und schauen, was so alles dazugekommen ist. Oder verschwunden, auch das kommt vor: So manche sinnvolle und wünschenswerte Einstellung hat Google da mit der Zeit wieder "wegoptimiert"...

Alle Eventualitäten kann ich hier natürlich eben so wenig berücksichtigen, wie jedes kleine Detail. Die folgende Kurzübersicht orientiert sich daher an den Systemeinstellungen, die Froyo (Android 2.2) anbietet – mit dem einen oder anderen Schwenk nach oben oder unten. In diese Einstellungen gelangt man, indem man vom Home-Screen ausgehend die "Menü-Taste" drückt, und im sich öffnenden Menü den Punkt "Einstellungen" (bzw. "Systemeinstellungen", "Settings", "System Settings" – verschiedene Hersteller bzw. Spracheinstellungen können natürlich leicht variieren) auswählt. Diese Stelle soll im Folgenden als Ausgangspunkt dienen – denn hier folgen die einzelnen "Sektionen", deren Name (jedenfalls meistens) aussagekräftig in Bezug auf die dahinter verborgenen Einstellungen ist:

Drahtlos und Netzwerke

Eindeutig: Hier funkt es. Zumindest kann man hier die entsprechenden Einstellungen tätigen. Mit den "Häkchen" (siehe rechtes Bild) lassen sich einzelne Features (de-)aktivieren. So sorgt der *Flugzeugmodus* für Funkstille: Keine mobilen Daten, kein Telefon – alles, was die empfindliche Bordelektronik stören könnte, wird damit auf einen Schlag abgeschaltet. Damit kann man den Androiden auch an Bord eines Flugzeugs nutzen – etwa um eBooks zu lesen. Handelt es sich um eine fortschrittliche Fluggesellschaft, die WLAN an Bord anbietet, so lässt sich letzteres auch im Flugzeugmodus separat aktivieren.

Hinter den Feldern mit dem auf der Spitze stehenden dreieckigen Symbol verstecken sich weitere Konfigurations-Bildschirme: So kann man unter "Wifi-Einstellungen" etwa in Reichweite befindliche WLAN-Netze anzeigen lassen, sich mit selbigen verbinden, gerade "abwesende" Netze manuell hinzufügen – oder auch entfernen. Spezielle Einstellungen pro Netzwerk sind, abgesehen vom ggf. benötigten Zugangsschlüssel, hier in der Regel nicht möglich.

Die Bluetooth-Einstellungen erlauben die Konfiguration der Sichtbarkeit des eigenen Gerätes, die Vergabe eines Gerätenamens, sowie das Scannen nach in Reichweite befindlichen sichtbaren Geräten – natürlich nur, sofern Bluetooth auch aktiviert ist.

Über die [VPN](#)-Einstellungen lassen sich die entsprechenden sicheren Verbindungen ins Firmen-, Uni- oder heimische Netzwerk erstellen und verwalten.

Zu guter Letzt lassen sich über den (im Screenshot nicht mehr sichtbaren) letzten Punkt *Mobilnetze* die Zugangspunkte ([APN](#)) für das mobile Netzwerk verwalten.



Anrufeinstellungen



Nomen est omen: Hier geht es um unsere Anrufe. Es lassen sich unter *Anrufbegrenzung* zugelassene Rufnummern verwalten, Einstellungen für die zu verwendende Mailbox (im Normalbetrieb sowie im Roaming separat) treffen. Man kann festlegen, ob bei abgehenden Gesprächen die eigene Nummer angezeigt werden soll, ein Profil für die Sprachqualität wählen, die Betreiberauswahl konfigurieren (automatisch aus verfügbaren Netzen auswählen, oder einen Betreiber fest einstellen). Auch um eventuelle Anrufweiterleitungen (ständig / bei besetzt / bei nicht-antworten / bei nicht-erreichbar) lassen sich hier einrichten bzw. verwalten – dazu gehört übrigens auch, ob und wann die Mailbox "rangehen" soll. Zu guter Letzt wäre da noch die Sache mit den "Anklopfern", die sich meist hinter dem Punkt *Zusätzliche Einstellungen* verbirgt.

Wer nun genau hinschaut sieht: Mit dem Screenshot habe ich ein wenig gemogelt, der stammt schon von Android 2.3 (aka "Gingerbread"). Was man einfach an einem zusätzlichen Konfigurations-Block erkennen kann: *Einstellungen für Internetanrufe*. Ab Gingerbread ist nämlich von Haus aus die Möglichkeit gegeben, mittels sogenannter **SIP**-Konten Telefonate über das Internet zu führen. Nachdem sich viele Netzanbieter dagegen gesperrt haben, bieten einige davon mittlerweile selbst entsprechende Dienste an. Zu den bekanntesten SIP-Anbietern zählen u. a. SIPGate, 1&1, Dus.NET und QSC.

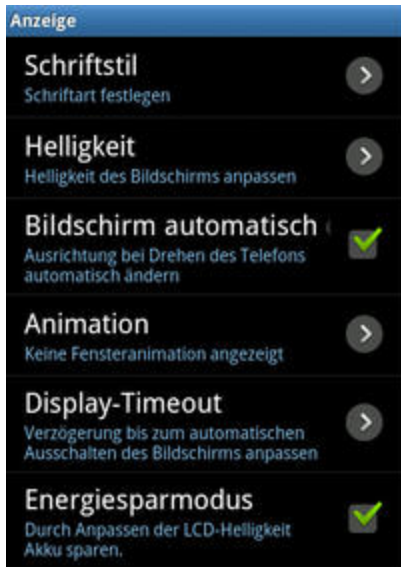
Töne

Ich höre schon die Beschwerden: "Das sieht bei mir aber ganz anders aus!" Wahrscheinlich war das bereits bei mindestens einem der vorigen Screenshots so, und wird auch bei weiteren wieder auftreten. Was daran liegt, dass jeder Hersteller ein wenig sein eigenes Süppchen kocht – und verschiedene Dinge anders einsortiert, oder erfreulicherweise zusätzliche Dinge zur Verfügung stellt. Beschränken wir uns also auf Prinzipielles, und ignorieren Abweichungen gutmütig.

Bei den *Toneinstellungen* geht es im Großen und Ganzen also um das, was es auf die Ohren gibt (ja okay: Auf das "haptische Feedback" trifft das natürlich nur zu, sofern das Gerät dabei direkt am Ohr befindlich ist). Separat lassen sich hier die verschiedenen Lautstärke-Einstellungen (Klingelton bei eingehenden Anrufen, Benachrichtigungen, Medien, Wecker...) vornehmen, oder das Telefon auch einfach "stumm" schalten. Der zu verwendende Sound bei eingehenden Anrufen lässt sich auswählen, man kann das "haptische Feedback" (vibrieren bei Betätigung von "Soft-Keys") sowie die Tastentöne bei der Telefonwahl einstellen, und mehr (oder auch nicht, je nach Hersteller). HTC bietet beispielsweise noch einen Taschenmodus (siehe Screenshot) – nicht für Billard, sondern für lauterer Klingeln, sollte sich der Androide in einer Tasche befinden (und dies auch bemerken). Motorola hat dafür einen Taschenmodus im Display-Menü (nächster Punkt), um selbiges beim Einstecken in die Hosentasche automatisch zu deaktivieren.



Display



Unter *Anzeige* (oder *Display*) finden sich die Einstellungen, die für den Bildschirm relevant sind – wie beispielsweise die Helligkeit, ob die Anzeige automatisch mit ins Querformat gedreht werden soll, wenn das Gerät gedreht wird, und ob Fenster-Animationen angezeigt werden sollen. Ebenso in diesem Menü zu finden ist das *Display-Timeout* – also der Zeitraum, nach dem der Bildschirm automatisch abgeschaltet werden soll (sofern der Benutzer "nichts gemacht" hat).

Weiteres variiert wieder nach Hersteller, und verschiedene [Custom ROMs](#) ergänzen auch gern das eine oder andere: Motorola möchte (zumindest beim Milestone 2) automatisch erkennen, ob sich das Gerät gerade in einer Tasche befindet (und bietet für diesen Fall die Option einer automatischen Bildschirm-Abschaltung – funktioniert aber in meiner

Hemdtasche schon Mal nicht). Dann gibt es Hersteller, die durch eine automatische Anpassung der Bildschirm-Helligkeit Akku sparen wollen, und anderes mehr.

Standort und Sicherheit

Wer jetzt meint, hier ginge es um Standort *und* Sicherheit, der irrt. Richtig heißen muss es: Standort *oder* Sicherheit. Spätestens ab Android 2.3 (Gingerbread) gibt es nämlich nur noch eins von beidem: Wer die (energiesparendere) Standortbestimmung über Mobilfunkmasten und WLAN-Netzwerke nutzen möchte, muss Google seine Standortdaten zur Verfügung stellen – ein Opt-Out bei gleichzeitiger Nutzung des Services ist nicht (mehr) vorgesehen (wie sich das dennoch bewerkstelligen lässt, findet sich beim Thema [Location-Cache](#)).

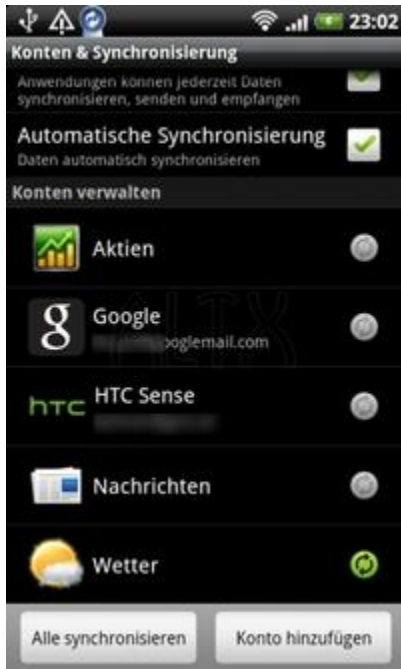
Daneben (oder der Anordnung nach besser "darunter") gibt es natürlich die Möglichkeit, die Standort-Bestimmung über [GPS](#) ein- bzw. auszuschalten. Sofern die andere Ortsbestimmungsart nicht aktiviert wurde, sammelt hier (hoffentlich?) keiner Daten mit – und solange nicht tatsächlich eine Ortsbestimmung durchgeführt wird, kostet das auch nicht wirklich Energie. Wenn aber doch, dann gleich richtig: Da kann es durchaus passieren, dass nach ca. vier Stunden Navigation der Akku aufgibt...

Passend zum Thema lässt sich bei manchen Geräten in diesem Menü auch der Kompass kalibrieren. Auch das Thema "Sicherheit" ist (entgegen meiner



eingehenden Feststellung) gelegentlich an diesem Ort untergebracht: Display-Sperre, SIM-Sperre, Geräteverwaltung (Einrichten/Verwalten von Geräteadministratoren), sowie Zertifikate lassen sich zumindest auf dem *Motorola Milestone 2* in diesem Menü finden.

Konten



Hier fällt die Länge der Liste zum Teil sehr unterschiedlich aus. Was zum Einen vom Hersteller und seiner "Zwangsbeglückung" zu tun haben kann (siehe "Aktien" im Screenshot eines HTC-Gerätes) – zum Anderen aber nicht zuletzt davon abhängt, wie viele "Konten" (z. B. Google-Accounts) man eingebunden hat. Natürlich spielt es auch eine Rolle, ob der Hersteller einen Geräte-Spion (HTC: Sense.COM, Motorola: Blur, etc.) zur Auffindung eines "verlorenen" Gerätes integriert, und der Anwender diesen eingerichtet hat und nutzt (im linken Screenshot: "HTC Sense").

Nach Auswahl des Google-Kontos ließe sich hier auch konfigurieren, ob Kontakte und Kalender automatisch synchronisiert werden sollen. Gut versteckt – und natürlich per Default aktiviert. Das "zentrale Häkchen" für die automatische Synchronisation, wie auf dem Screenshot zu sehen, fehlt beispielsweise bei oben genanntem *Motorola Milestone 2* an dieser Stelle ebenso, wie die

Möglichkeit, die Hintergrund-Synchronisation zu deaktivieren (beide "Häkchen" sind bei HTC-Geräten hier vorhanden) – auch an anderer Stelle konnte ich sie bislang nicht finden. Wie gut, dass es für so etwas [Schnellumschalter](#) gibt...

Anwendungen

Anwendungen konfigurieren? Nein, vielmehr geht es um deren Verwaltung. Werden beispielsweise die *unbekannten Quellen* nicht aktiviert, ist eine Installation nur über den Playstore möglich – um also eine App aus einer vom Entwickler zur Verfügung gestellten [APK-Datei](#) installieren zu können, muss das Häkchen hier zwingend gesetzt werden.

Unter *Anwendungen verwalten* werden alle installierten Apps aufgelistet. So kann man hier nachschauen, wie viel Platz die Apps selbst belegen, wie viele Daten sie verwalten, und wie groß der vorgehaltene Cache gerade ist. Und man kann all diese Dinge auf einen Schlag oder auch einzeln



löschen – wobei das Löschen einer App auch automatisch das Löschen ihrer Daten und ihres Caches nach sich zieht, nicht aber umgekehrt. Dies ist beim Playstore manchmal ganz hilfreich, wenn der sich gerade wieder verhaspelt hat: Cache löschen, neu initialisieren lassen.

Hinter den *Ausgeführten Diensten* verbirgt sich ein Mini-Task-Manager. Dieser zeigt u. a. an, welche Apps gerade ausgeführt werden, seit wann, und wie viel Speicher sie belegen. Letzteres sowohl einzeln, als auch in Summe. Ebenfalls an dieser Stelle zu sehen: Welche Apps welche Services gestartet haben. Der bekannteste Übeltäter dürfte da Google Maps mit seinem *NetworkLocationService* sein – läuft ständig, auch wenn man Maps nie genutzt hat (wird vermutlich für die netzbasierte Standortbestimmung genutzt – doch sicher sagen kann das scheinbar niemand). Durch Antippen eines Eintrags erhält man die Möglichkeit, einen Dienst anzuhalten – oder auch nicht. Das "oder auch nicht" bezieht sich zum Einen auf die Möglichkeit, diese Aktion abzubrechen; zum Anderen aber auch auf den Fakt, dass die meisten Dienste ohnehin nach wenigen Sekunden wieder gestartet sind (worauf der Dialog auch hinweist).

Dann wäre da noch der Punkt *Entwicklung* – nicht nur für Entwickler. Hier verstecken sich auch Dinge wie "Display aktiv lassen, wenn Ladekabel angeschlossen".

Weitere Punkte sind wiederum Hersteller- und Geräte-abhängig – wie etwa der *Schnellstart* bei neueren HTC-Geräten, oder die Frage, was bei "2x Home" gestartet werden soll (Milestone 2).

Datenschutz



Schon wieder etwas Doppeldeutiges: Möchte Google hier unsere Daten schützen (indem es sie auf seine Server sichert) – oder sollen wir hier unsere Daten vor Google schützen, indem wir alle Häkchen entfernen? Sollte letztere Variante die richtige sein, ist die Umsetzung nur teilweise gelungen – denn K&K (Kalender und Kontakte) ignorieren diese Einstellung geflissentlich, und synchronisieren trotzdem...

Scherz beiseite: Gemeint ist, dass die eigenen Einstellungen sowie Anwendungsdaten auf die Server von *Google Inc.* gesichert werden bzw. nach einem Werksreset von dort wieder hergestellt werden sollten. Warum der Konjunktiv an dieser Stelle? Dieser Service funktioniert leider nicht immer so zuverlässig, wie man es sich wünschen würde. So liest man im [Bug-Tracker](#) von etlichen Fällen, in denen Apps und Daten nur teilweise oder überhaupt nicht wieder hergestellt wurden...

Und was wird hier überhaupt gesichert? Für Geräte, auf denen Android 3.x oder höher eingesetzt wird, finden sich Details dazu im [Benutzer-Handbuch](#). Demnach werden etwa folgende Daten gesichert (und hoffentlich bei Bedarf auch wieder hergestellt):

- Android-Einstellungen wie WLAN-Netzwerke mit ihren Passwörtern, das Benutzer-Wörterbuch, etc.
- Einstellungen vieler Google-Apps, einschließlich der Lesezeichen des Web-Browsers
- Die aus dem Playstore installierten Apps
- Daten von anderen Apps, sofern die Entwickler dies explizit implementiert haben

Es handelt sich hierbei also mitnichten um ein vollständiges Backup: Daten von Apps werden nur dann mitgesichert, wenn diese Apps das auch explizit unterstützen – was für viele Apps nicht zutreffen dürfte. Und natürlich sind auch die auf der SD-Karte abgelegten Dateien nicht in diesem Backup enthalten. Was bedeutet, dass man sich zumindest nicht auf dieses Backup allein stützen sollte. Auch ist es nicht jedermanns Ding, seine Daten einer Cloud anzuvertrauen – auch wenn dies (Achtung!) die Voreinstellung ist.

Der Werksreset ist übrigens ein weiterer Punkt, der sich auf vielen Geräten in diesem Menü findet: "Zurücksetzen auf Werkseinstellungen". Entweder, wenn gar nichts mehr geht und alles nur noch verrückt spielt – oder wenn man das fragliche Gerät veräußern möchte, denn bei einem "Werksreset" werden alle Anwenderdaten einschließlich der vom Anwender installierten Apps gelöscht. Lediglich die eventuell vorhandene SD-Karte bleibt dabei völlig unberührt (abgesehen von App2SD)...

SD-Karte und Telefonspeicher

Wenn jemand seinen Speicherplatz sucht: Hier wird derjenige fündig. Dass der Inhalt auf dem Screenshot so umfangreich ist, liegt am Gerät: Das *Samsung Galaxy S* hat zusätzlich zu "Telefonspeicher" und "externer SD-Karte" auch noch eine "interne SD-Karte" (richtiger gesagt: Die "interne Karte" wurde in "Telefonspeicher" und "interne SD-Karte" aufgeteilt – von SD-Karten mit 6 Gigabyte Speicher habe zumindest ich noch nichts gehört oder gelesen; nimmt man jedoch den Telefonspeicher dazu, kommt man auf vertraute 8 Gigabyte...).

Wie dem auch sei: Hier findet man heraus, wie viel Speicher insgesamt vorhanden und wie viel davon verfügbar ist – sowohl auf Karten, als auch im Telefonspeicher. Darüber hinaus lässt sich die SD-Karte aus diesem Menü heraus formatieren, oder auch "deinstallieren" (unter Windows heißt das "sicher entfernen"; gemeint ist, das System auf die physikalische Entfernung aka Entnahme der SD-Karte vorzubereiten).



Sprache und Tastatur



In diesem Menü geht es um die Lokalisierung. Das betrifft zum Einen Spracheinstellungen, nach denen sich auch die Apps (sofern entsprechend angepasst) richten sollen – damit man auch versteht, was man da liest. Durch das zugehörige Land werden darüber hinaus auch Dinge wie das Datums- und Uhrzeitformat, sowie das zu verwendende Einheiten-System (metrisch oder imperial) beeinflusst.

Zum Zweiten geht es aber auch um die zu verwendende Tastatur ("Eingabe-Methode") und deren Konfiguration. Wie am Screenshot erkennbar, ist beim *Milestone 2* die Tastatur-App *Swype* vorinstalliert – und lässt sich daher hier auch konfigurieren. Und zwar inklusive der von ihr unterstützten Sprachen, die sich im laufenden Betrieb sodann durch ein Wischen über die Leertaste wechseln lassen. Die im Screenshot angezeigte Option *WiFi Keyboard* hingegen ist nicht von Haus aus dabei: Hier handelt es sich um eine Ergänzung der App [Remote Web Desktop](#). Aktiviert man diese, kann man die PC-Tastatur verwenden, um Texte am

Androiden zu erfassen. Unter *Tastatur des Geräts* wiederum ist die Hardware-Tastatur des *Milestone 2* zu verstehen.

Der letzte Punkt ist das *Nutzerwörterbuch*, welches die selbst hinzugefügten Begriffe für die automatische Worterkennung enthält. Hier lassen sich neue Wörter direkt hinzufügen – aber auch eventuell falsch/versehentlich erfasste Wörter wieder entfernen.

Spracheingabe/-ausgabe

Dem Screenshot nach ein kurzes Menü – doch hinter den beiden dargestellten Punkten verbirgt sich eine wahre Flut an Einstellungen. Der erste Punkt bedenkt dabei die *Google-Spracherkennung*: Welche Sprache wird gesprochen? Mit welchen Einstellungen soll die Suchoption *SafeSearch* versehen werden? Sollen Beleidigungen automatisch gesperrt, sollen im Suchfeld Hinweise eingeblendet werden?



Ähnlich sieht es dann für die Sprachausgabe, auch TTS (Text-To-Speech) genannt, aus. Für diese lassen sich Sprache, Stimme, Sprechgeschwindigkeit, Sprachmodul und gegebenenfalls auch Sprachdaten konfigurieren bzw. installieren.

Eingabehilfen



Was zum Geier sind denn nun Eingabehilfen – Tastatur und Spracheingabe hatten wir doch schon? Derartige "Eingabehilfen" sind hier auch nicht gemeint; dieser Punkt richtet sich vielmehr an Menschen mit Behinderungen. Da wäre zuerst die *Sprachwiedergabe*: Hat der Benutzer so schlechte Augen, dass er zwar den Text sieht, ihn aber nicht lesen kann, kann er ihn sich durch "Antippen" vorlesen lassen. Ähnlich verhält es sich mit dem *Zoom-Modus* zur Vergrößerung von Elementen.

Dass *Tasker* hier auftaucht, hat wohl eine andere Bewandnis: Der braucht die entsprechenden Berechtigungen, um im "Benutzer-Interface" (also auf dem Bildschirm) dargestellte Elemente auswerten zu können.

Einfach und verständlich hingegen ist der letzte dargestellte Punkt: *Ein/Aus beendet Anruf*. Gut für alle die, die den entsprechenden Button auf dem Bildschirm nicht finden können: Statt das Display auszuschalten, wird dann per Ein/Aus-Taste ein laufendes Gespräch beendet. Natürlich nur, wenn eines läuft. Das hat bei manchen den dummen Nebeneffekt, dass sie mitten im Telefonat versehentlich auflegen – weil sie zum "eben Mal etwas nachschauen" das Display anschalten wollen...

Datum und Uhrzeit

Dieses Menü sollte auch einfach und verständlich sein – eigentlich erübrigt sich hier fast jede Erklärung. Hier lässt sich bei Bedarf die Zeit manuell korrigieren, falls das Netzwerk des Betreibers das nicht hinbekommt; normalerweise sollte dies recht selten bis nie nötig werden. Einzig die Umschaltung vom 12- auf den 24-Stunden-Modus (oder umgekehrt) sowie die Anpassung des Datumsformates kann, je nach Vorliebe, für den Einen oder die Andere interessant werden.



Telefoninfo



Das letzte Menü birgt eine Menge Potential. Zuallererst finden sich hier die vom Betreiber bereitgestellten Systemsoftware-Updates – für all die Ungedulden, die nicht auf die entsprechende automatische Benachrichtigung warten können oder wollen. Dies ist der einzige Punkt, in dem auch eine Aktivität ausgelöst wird: Das Einspielen eines Updates, so denn eines gefunden wird.

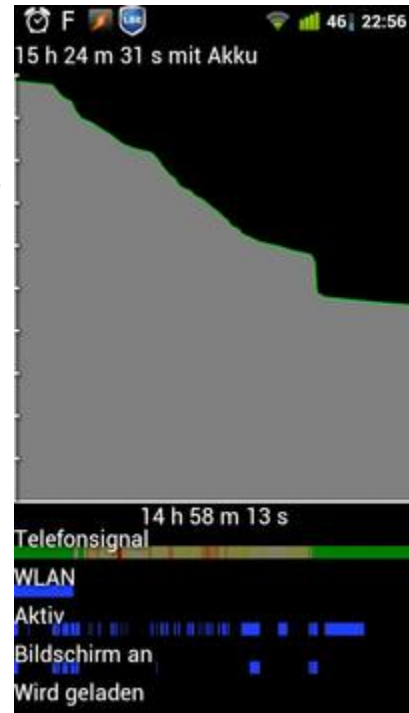
Alle anderen Punkte spucken hingegen jede Menge nützlicher Informationen aus – und sind auf verschiedenen Geräten unterschiedlich zusammengefasst. Sie informieren über Akkustatus (Laden/Entladen) und -stand, die eigene Telefonnummer, das aktive Netzwerk (Provider) sowie dessen Signalstärke und Typ, Roaming, Mobilnetzstatus, IMEI, MAC-Adressen und

Betriebszeit und geben schließlich diverse Informationen zum verwendeten System.

Unter dem Punkt *Akku* (der beispielsweise auf dem *Motorola Milestone 2* komplett aus diesem Menü genommen und in ein eigenes verfrachtet wurde) findet man in der Regel auch eine Liste der größten Verbraucher. Diese scheint nach meinem Eindruck jedoch mit fortschreitender Android-Version immer "Allgemeiner" zu werden – vielleicht täusche ich mich da aber auch...

Denn mit Gingerbread kam zumindest ein neues Feature hier hinzu: die rechts abgebildeten Statistiken zum Batterieverbrauch. Zu diesen gelangt man, wenn man im Menü *Akku* auf den "Balken" am oberen Bildschirmrand tippt – der sich bei genauerem Hinschauen als Mini-Graph entpuppt, wie der "große Graph" auf dem Screenshot deutlich macht.

Während dieser "große Graph" veranschaulicht, wie (un)gleichmäßig der Akku entladen wurde, finden sich unmittelbar darunter einige wertvolle Details. So beispielsweise zum Telefonsignal: Ein sattes Grün steht für eine sehr gute Signalstärke; wird es heller, war das Signal schwach. Kritisch wird es bei roten Stellen: Hier war gar kein Signal vorhanden – das Smartphone hat aber mit aller Gewalt versucht, eines zu finden. Im Luftschutzbunker relativ sinnlos: Wenn es kein Signal gibt (etwa aufgrund starker Stahlbeton-Wände, oder weil man sich etwa irgendwo abseits jeglicher Zivilisation befindet), bedeutet dies nur eins: Hoher Akku-Verbrauch, da das Gerät mit voller Leistung auf (nutzlose) Suche geht. Tritt dies jeden Tag zur gleichen Zeit auf, sollte man die zugehörige Örtlichkeit prüfen – und ggf. per [Automat](#) hier in den Flugzeugmodus wechseln.



Die übrigen vier Graphen sind eigentlich selbsterklärend: Wann war WLAN aktiviert, wann das Gerät beschäftigt, wann der Bildschirm eingeschaltet, und wann das Ladekabel angeschlossen (natürlich beidseitig).

Ausgewählte bzw. häufig genutzte Einstellungen

Schnell und ohne große Umstände an die wichtigsten Einstellungen gelangen – darum geht es hier. Sofern ein pures Umschalten im Sinne von "An/Aus" gefragt ist, haben wir dieses Thema ja bereits bei den [Schnellumschaltern](#) abgehandelt. Doch nicht alles ist dort enthalten. Daher schauen wir nach, was es noch so gibt.

"Schnelle Einstellungen" – auf Englisch hieße das "Quick Settings". Und mindestens vier Apps tragen genau diesen Namen. Die bestbewertetste App davon ist [Quick Settings](#) von Sergej Shafarenka (rechtes Bild): durchschnittlich 4,7 Sterne bei fast 35.000 Bewertungen sind schon allerhand! Die App verankert sich, wie am Screenshot rechts ersichtlich, mit einem Symbol in der Statuszeile. Von dort gelangt man in dargestelltes Menü, und findet die wichtigsten Schalterchen vor. Über den kleinen Knopf neben "Ringer" erreicht man eine Reihe Schieberegler für die verschiedenen Lautstärken (Klingelton, Benachrichtigung, Medien, etc.).

Zwei Gimmicks sind auch noch zu erkennen: Die Taschenlampen-Funktion und, etwas versteckt, die Liste der Stromverbraucher. Letztere öffnet sich, wenn man das Batterie-Symbol antippt.





Gleicher Name, andere App: [Quick Settings](#) von APS (linkes Bild). Auch hier ein recht ähnliches Popup, auf welches man auf verschiedene Art zugreifen kann: Über die Statusleiste, langes Drücken auf die "Such"-Taste, oder langes Drücken auf die Kamera-Taste – frei nach Gusto, wie man es konfiguriert hat. Neben einer ganzen Reihe von Schnellumschaltern finden sich auch bei dieser App die Schieberegler für die Lautstärke. Als praktisches Extra gibt es eine Übersicht über wichtige Systemdaten: Genauer Akkustand, Anzahl installierter Apps, sowie Gesamter und verfügbarer Speicher (intern und SD-Karte).

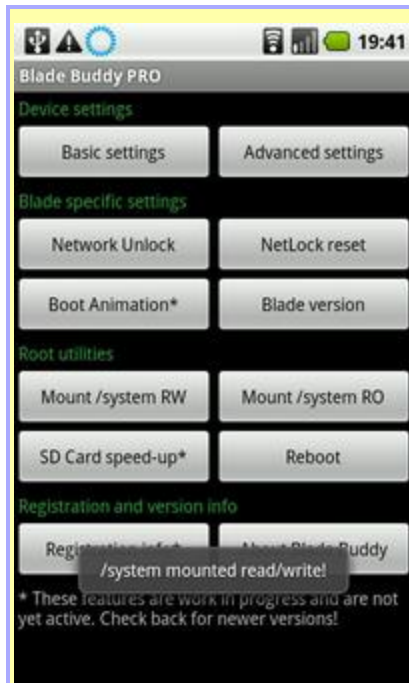
Da die anderen mir bekannten "Schnelleinsteller" ähnlich aufgebaut sind, möchte ich sie nicht alle einzeln vorstellen. Stattdessen kann sie jeder bei Bedarf in [dieser Übersicht](#) selbst nachschlagen.

Zusätzliche bzw. versteckte Einstellungen

Wenn es um zusätzliche Konfigurations-Möglichkeiten geht, die das System von sich aus in der Form nicht anbietet, fällt den Meisten sicher als erstes *Spare Parts* ein. Oder besser noch, da aktueller und ohne Werbung: [Spare Parts Plus](#). Diese App stellt nicht nur einzelne zusätzliche Einstellungen zur Verfügung, die nicht selten wesentlich mehr ins Detail gehen als die Bordmittel (beispielsweise Animationen, siehe Screenshot: Nicht nur ein simples globales "An/Aus", sondern explizite Einzelheiten). Anwender mit "schlechten Augen" werden sich zudem über die Möglichkeit freuen, die Größe der Systemschrift anpassen zu können. Und darüber hinaus versorgt uns *Spare Parts Plus* auch noch mit etlichen detaillierten System-Statistiken, wie ebenfalls auf dem Screenshot erkennbar.

Die App ist so gut, dass sich das Team des bekannten [Custom-ROMs](#) [CyanogenMod](#) entschlossen hat, es von vornherein in seine ROM einzubauen. Und wem die App gut gefällt, der kann für knapp anderthalb Euro auch zur Pro-Version greifen.





Wer seinen Androiden mit [root](#)-Rechten ausgestattet hat, greift (zusätzlich? stattdessen?) vielleicht eher zu [Blade Buddy](#) (linkes Bild) oder dessen Pro-Version, beide aus dem gleichen Hause wie *Spare Parts*. Die App integriert sich in *Spare Parts* (bzw. lässt sich aus *Spare Parts* heraus aufrufen), und erlaubt die Konfiguration etlicher systeminterner Schalterchen.

Der Screenshot zur linken gibt einen ganz groben Überblick über die bereitgestellten Funktionalitäten. So lässt sich das Auslösegeräusch der Kamera abschalten, die Dalvik-VM detailliert anpassen, oder auch die Boot-Animation deaktivieren. Aktionen wie ein schneller Reboot, oder das beschreibbare Einhängen der Systempartition stellen die App ebenfalls vor keinerlei Probleme.

Natürlich wird vor all zu wildem "Gespiele" gewarnt. Einige dieser Einstellungen hat Google nicht umsonst vor dem "Otto Normalanwender" versteckt, und für nicht-rootler unzugänglich gemacht: Das falsche Schraubchen zu weit gedreht, und da fliegt das Blech weg. Man kann sich sein System auch kaputt-konfigurieren – also bitte mit Bedacht vorgehen!

Aber benötigt man zum Sichtbarmachen "versteckter" Einstellungen nun wirklich immer extra Apps? Oder gibt es da auch andere Möglichkeiten, an diese zu gelangen? In der Tat gibt es für so manche "Secret Settings" auch passende "[Secret Codes](#)" – wie beispielsweise `*##4636##`. Diese gibt man so ein, als wollte man bei der entsprechenden Nummer anrufen. Und in diesem Beispiel öffnet sich dann ein recht informatives Menü, welches auch *Spare Parts* und [Secret Phone Settings](#) (rechtes Bild) nutzen. Wobei bei letzterer App das dauernde "Info" auf den Buttons leicht irritiert: Ja, es werden etliche hilfreiche Informationen angezeigt – dabei lässt sich aber durchaus auch die eine oder andere Einstellung anpassen (siehe Screenshots auf der Playstore-Seite).

Neben diesen Tools für vielfältige allgemeine Einstellungen gibt es jedoch auch noch einige, die sich auf Einzelbereiche spezialisiert haben. So lassen sich etwa mit dem [WiFi Advanced Config Editor](#) für jedes konfigurierte WLAN erweiterte Einstellungen wie der Index des WEP-Schlüssels oder WPA Enterprise Parameter (z. B. `wpa_supplicant`) anpassen, mit [Set DNS](#) zu verwendende NameServer und andere [DNS](#)-Parameter konfigurieren, und ([root](#)-Rechte vorausgesetzt) sogar mit [ProxyDroid](#) die Verwendung eines [Proxy-Servers](#) (auch abhängig von der Verbindungsart: mobil oder WLAN) erzwingen.



Nicht ganz uninteressant wäre auch die Möglichkeit, die Synchronisation von Kalender, Kontakten, Google Mail & Co. auf die Zeiten zu beschränken, zu denen man mit einem WLAN und/oder einer Stromquelle verbunden ist - [SmartSync](#) macht dies möglich. Root-Usern vorbehalten bleiben allerdings Dinge wie die Anpassung der CPU-Taktung (das bekannteste Tool hierfür wäre sicherlich [SetCPU](#), aber auch der [CPU-tuner](#) ist definitiv einen Blick wert!) oder der Bildschirm-Auflösung (z. B. mittels [LCDensity](#) - hier sieht man an den Beispiel-Screenshots auch wunderbar die Auswirkungen).

Und wer jetzt noch mehr Möglichkeiten oder Alternativen sucht, wird in [dieser Übersicht](#) fündig.

Automatisierung

[Einstellungen](#) sind ja gut und schön. Einmal gemacht, merkt sie sich unser Android auch brav. Aber passen sie auch zu jeder Gelegenheit? Tagsüber auf der Baustelle muss ein eingehender Anruf möglichst laut signalisiert werden, damit man ihn auch mitbekommt. Doch wer lässt sich schon gern nach Feierabend etwa beim Schmusen von einem Donnerwetter in der Hose (oder wo immer das Gerät gerade steckt) aufschrecken? Genauso unpassend ist ein lautes Klingeln etwa während eines Seminars, oder im Konzertsaal (auch wenn bei heutiger "moderner Musik" das halbe Publikum denken könnte, das gehöre dazu).

Und das ist erst der Anfang: Das Umschalten zwischen verschiedenen Klingeltönen und Lautstärken vergessen wir eben so leicht, wie das Aktivieren von GPS vor dem Start der Kartensoftware – oder das Hochschrauben des Display-Timeouts vor Aktivierung des RSS- bzw. eBook-Readers. Wie viele andere Szenarien ich jetzt hier nicht einmal angeschnitten habe, will ich dabei gar nicht mutmaßen...

Für all diese Dinge muss es doch auch Helferlein geben? Wir sind doch sicher nicht die Ersten, die vor diesem Problem stehen! – Das ist beides korrekt. Und so gibt es Apps, die uns gewisse Dinge abnehmen können: [Zeitgesteuert](#), [abhängig vom aktuellen Standort](#) ("location-based" oder "ortsbasiert" genannt), oder auch [kombiniert bzw. bei anderen Events](#). Und es gibt [Tasker](#). Der passt in keine dieser Kategorien hinein: Tasker ist eher ein "Wundertool", "Schweizer Offiziersmesser", oder einfach Das Ultimative Werkzeug (Englisch: "TUT", **The Ultimate Tool**). Braucht aber für Einsteiger auch mindestens ein "Tut" (Tutorial) – doch dazu später...

Ein Tipp noch vorweg: Automatisierungs-Tools (die übrigens in [dieser Übersicht](#) zusammengefasst sind) sollten möglichst nicht auf der SD-Karte installiert werden. Sonst werden sie beendet, wenn man seinen Androiden per USB mit dem Computer verbindet...

Zeitschaltuhren

Geht es um reines zeitbasiertes Schalten, ist [Timeriffic](#) (rechtes Bild) nicht nur das bekannteste, sondern auch das am besten bewertete Tool. Ein Großteil der oben genannten Aufgaben lässt sich damit erledigen – sofern es mit klassischen Profilen zu tun hat: Lautstärken, Stummschaltung, Vibrator. Auch Flugzeugmodus, WLAN, und Bluetooth beherrscht die App. All diese Dinge lassen sich zu beliebigen Zeitpunkten auf beliebige Werte stellen – was übrigens auch für die Bildschirm-Helligkeit gilt.

Übrigens nicht nur Uhrzeit-basiert: Auch die Wochentage lassen sich dazu auswählen. Der Screenshot zeigt das ganz gut: So lassen sich etwa für das Wochenende andere Regeln erstellen, als man sie werktags einsetzt – denn wer möchte schon sonntags früh um sieben Uhr von einem lauten Klingeln geweckt werden.

Das war es aber auch bereits – mehr ist nicht drin. [Android Audio Profile](#) bietet etwa das gleiche an (mit der zusätzlichen Möglichkeit, eine Schaltung per Widget vornehmen zu können). Und damit ist dann – für die reinen Zeitschaltuhren – auch schon das Ende der Fahnenstange erreicht.



Ortsbasiertes Schalten



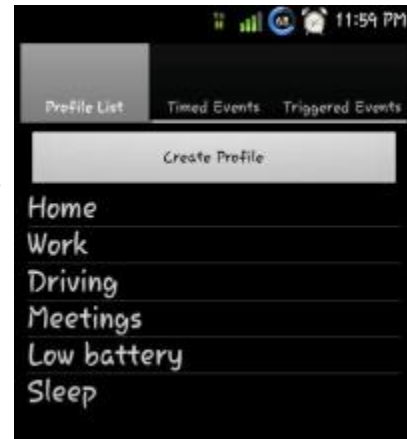
Hier gibt es eigentlich nur zwei Kandidaten: Das auf die übrigen klassischen Profile beschränkte [Location Profile](#), welches sich lediglich um Lautstärke, Klingeltöne, Vibration sowie Hintergrundbilder ("Wallpaper") kümmert – und [Llama](#) (linkes Bild).

Llama gibt sich vielfältiger. Abhängig von den jeweiligen Standorten (der Reiter *Areas* im Screenshot), kann sich die App neben genannten klassischen Audio-Profilen auch um Dinge wie [APNs](#), Hintergrundbilder, WLAN, und den Flugzeugmodus kümmern – oder aber den Androiden neu starten, Apps starten/beenden, und mehr. Und eigentlich muss sich *Llama* langsam einmal Umbenennen, und gehört sodann in die nächste Kategorie verschoben. Denn mittlerweile lassen sich auch andere Events zum Schalten heranziehen: Angeschlossene Ladekabel, Docking-Stations, Kalender-Ereignisse, in der Nähe befindliche Bluetooth-Geräte...

Womit die App fast mit *Locale* (siehe unten) gleichauf zieht – nur dass es kein GPS benutzt, und die entsprechenden Plugins nicht unterstützt.

Zeit-, Orts- und Eventbasiertes Schalten

Der Name verrät bereits, was sich mit **Phone Profiles** (rechts) schalten lässt: Die klassischen Telefon-Profil-Eigenschaften. Hinzu kommen noch WLAN, Bluetooth, der Flugzeugmodus, sowie Bildschirm-Helligkeit und Ausrichtungsmodus. Ortsbasiertes Schalten unterstützt die App zwar nicht – dafür aber neben zeitbasiertem Schalten eine ganze Reihe Events: Etwa ein angeschlossenes Ladekabel, oder ein bestimmter Akkustand. Damit lassen sich energiehungrige Aktionen einschränken – wobei ich beispielsweise an ein Dimmen des Displays bei niedrigem Akkustand, oder an das Abschalten der WLAN-Funktionalität denke.



Mindestens genauso praktisch ist das Schalten anhand bestimmter Kalendereinträge: Bei Sitzungen klingelt es nicht mehr, und auch beim Theaterbesuch ist es ruhig. Leider lässt sich derzeit wenig zur Zuverlässigkeit von *Phone Profiles* sagen, ohne es selbst auszuprobieren: Es gibt im Playstore noch keine aussagekräftigen Kommentare.



Das sieht bei **EasyProfiles** (linkes Bild) ganz anders aus: Hier finden sich zahlreiche, überwiegend positive Bewertungen einschließlich Kommentaren. Sowohl im Google Playstore, als auch bei AndroidPIT. Es scheint mir recht eindeutig: Diese App hat dem einstmaligen Spitzenreiter *Locale* den Rang abgelassen. Sogar die zahlreichen *Locale*-Plugins werden hier unterstützt, und *EasyProfiles* selbst kostet bei vergleichbarem Leistungsumfang mit ca. dreieinhalb Euro gerade einmal die Hälfte!

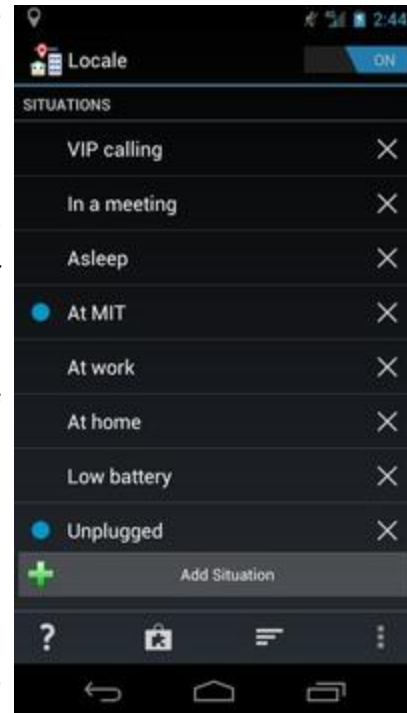
Hervorragender Support und übersichtliche Bedienung reichen bei *EasyProfiles* hoher Flexibilität die Hand. Neben automatischen Aktionen lassen sich Profile sogar manuell aktivieren. Die App ist in der Lage, zahlreiche Konfigurationsänderungen vorzunehmen: Alles, was zuvor genanntes *Phone Profiles* schalten kann, und zusätzlich auch Synchronisations-Einstellungen (AutoSync, Hintergrunddaten), APN, 2G/3G/4G, Display (Helligkeit, Timeout, Wallpaper, AutoRotate), und mehr.

Als Auslöser kommen neben Zeit und Ort auch Dinge wie Kalendereinträge, Akku-Stand, Bluetooth/WLAN Verbindungen, Dockingstation/Ladekabel und sogar die im Vordergrund befindliche App in Frage. Da sich all dies noch zusätzlich durch Verwendung von *Locale*-Plugins erweitern lässt, haben wir bei dieser App alle eingangs genannten Szenarien abgedeckt.

Nicht unerwähnt bleiben darf natürlich der Klassiker in dieser Rubrik: Lange Zeit galt **Locale** (rechtes Bild) als die einzige echte Lösung. Und was war davor?

*Judge Robert Restaino **jailed 46 people** when a mobile phone rang in his New York courtroom and no one would admit responsibility. So we invented **Locale**. Problem solved. (Im Mai 05 hat Richter Restaino 46 Leuten eine Haftstrafe verordnet, da keiner zugeben wollte, dass es sein Telefon war, welches im New Yorker Gerichtssaal klingelte. Da erfanden wir **Locale**. Problem gelöst.)* So ist es auf der **Website** zu lesen. Vergleichsweise einfach zu erfassen und bedienen, wäre **Locale** auch heute noch im Ranking weiter vorn – gäbe es da nicht das Handicap des Preises: Gut sieben Euro sind im App-Market einfach ein wenig viel, da schaltet so mancher bereits ab, ohne weiter zu lesen.

Locale bietet im Prinzip alles, was auch **EasyProfiles** im Portfolio hat. Ob die Bedienung hier intuitiver oder einfach nur vergleichbar ist, muss jeder für sich entscheiden. Ich habe **Locale** eine Zeit lang benutzt und sehe eigentlich nichts, was **EasyProfiles** zum halben Preis nicht bieten würde – abgesehen vielleicht von der längeren Erfahrung, die das **Locale**-Team mitbringen dürfte. Wobei ich nicht sagen kann, ob das geniale Prinzip des "Schichten-Aufbaus" dort auch übernommen wurde: Bei **Locale** bauen die Profile nämlich aufeinander auf. Das "unterste" Profil bildet die "Basis", und die "darüber liegenden" Profile wenden lediglich davon abweichende Einstellungen an. Wird also "weiter oben" ein Profil inaktiv, tritt automatisch der aus den "darunter liegenden" aktiven Profilen bestimmte Zustand ein – wobei die Eigenschaften der "weiter oben liegenden" jeweils Vorrang haben. So lässt sich jederzeit leicht erkennen, was Sache ist.



Tasker

Ein eigenes Kapitel für eine einzige App sei ungerechtfertigt? Dem kann ich in sofern zustimmen, als dass man **Tasker** ein eigenes Buch widmen könnte: "Android-Automation mit Tasker". Vielleicht mache ich das ja sogar einmal. *Tasker* ist nicht irgend eine Automatisierungs-App – es ist *die* Automatisierungs-App schlechthin. Ist etwas mit Tasker nicht hinzubekommen, dann geht es schlicht und ergreifend nicht. Auch nicht mit einer anderen App. Selbst im Forum findet sich kaum eine Frage, auf die nicht irgend jemand mit "Tasker" geantwortet hätte...

Tasker kann alles das, was auch *Locale* kann, einschließlich der Verwendung der Plugins. Ach was: *Tasker* kann alles, was irgend eine der vorgenannten Apps kann. Und das in beliebiger Kombination. Und mehr. Viel mehr!



Auch bei *Tasker* erstellt man "Profile". Dargestellt werden diese in der Übersicht etwa so, wie man es auf dem rechten Bild erkennen kann – zweispaltig: Links der (bzw. die) Auslöser ("Kontext" genannt), rechts das, was passieren soll ("Tasks", also Aufgaben). Bereits an diesem einen Screenshot lässt sich erahnen, wie vielfältig *Tasker* einsetzbar ist: Als Auslöser lassen sich neben den bisher besprochenen "gewöhnlichen" Dingen wie Zeit, Tag und Ort etwa Apps im Vordergrund, Hardware/Software Status, Ereignisse, Widgets, oder Timer einsetzen. Auch der Erhalt einer SMS mit einem bestimmten Text, oder der Anruf von einer bestimmten Telefonnummer sind möglich.

Ebenso vielfältig schaut es bei den Möglichkeiten für Tasks aus. Aufgrund ihres Umfangs mussten diese bereits in grauer Vorzeit auf Kategorien verteilt werden (linkes Bild; die Zahl besagt, wie viele Elemente in einer Kategorie enthalten sind). Außerdem lassen sich in Tasks auch Schleifen, Variablen und Bedingungen verwenden.

Im linken Bild sieht man bereits einen Eintrag "Plugins": Ja, auch Tasker ist in der Lage, mit den zahlreichen Locale-Plugins zu arbeiten. Darüber hinaus bieten auch immer mehr Apps direkte Schnittstellen für Tasker: So lässt sich etwa bei *FTPSyncX* (im Kapitel [lokale Dateisynchronisation](#) vorgestellt) eine Synchronisation anstoßen oder eine Einstellung ändern. Auch für *Dropbox* gibt es diverse Plugins. Und nicht zuletzt lassen sich bei Bedarf sehr maßgeschneiderte Lösungen wie etwa ein [Diebstahlschutz](#) oder ähnliches realisieren.

Wenn *Tasker* also das Non-plus-Ultra ist, warum nutzt es dann nicht jeder? Dafür gibt es sicher mehrere Gründe. Die Einen schreckt der Preis ab: Mit etwa fünf Euro ist die App nicht ganz billig – wenn auch, an Funktionsumfang und gebotenen Möglichkeiten gemessen, absolut nicht teuer. Wer jedoch nur einen kleinen Teil dieser Fähigkeiten benötigt, die zudem noch von einem anderen (vielleicht gar kostenlosen) Tool abgedeckt werden – dem kann man es ja wohl kaum verdenken, wenn er sich dann auch für letzteres entscheidet.

Ein weiterer Punkt ist die mit der App verbundene Komplexität: Wenn jemand behauptet, er hätte sich "mal eben *Tasker* installiert und vollständig eingerichtet" – so hat dieser Mensch glatt gelogen. Selbst jemand, der *Tasker* bereits "kennt wie seine Westentasche", macht dies nicht "mal eben so nebenbei". Zugegeben: Je besser man sich damit auskennt, je mehr setzt man damit auch um. Und hätten unsere Androiden einen Temperatur-, Wasser- *und* einen Kaffeesensor, könnte *Tasker* mit gerooteten Geräten auch Kaffee kochen (Kontext: Wasser < 90°C mit Kaffeepulver; Task: Übertakte CPU, starte alle Benchmarks. Exit-Task: Benchmarks stoppen, CPU wieder normalisieren)...

Zugegeben: *Tasker* mag ein wenig gewöhnungsbedürftig sein, und fordert zumindest ein wenig Einarbeitung. Aber auch wenn gelegentlich behauptet wird, man bräuchte dafür ein Diplom: So kompliziert ist es nun wirklich nicht. Und mir ist niemand bekannt, der den kleinen Aufwand zu Beginn später bereut hätte. Um neuen Anwendern den Einstieg zu erleichtern, habe ich im AndroidPIT-Forum eine kleine [Sammlung der wichtigsten Ressourcen](#) erstellt – hier finden sich Links zu Tutorials, Wikis und weiteren Hilfsquellen.

Übrigens: *Tasker* gibt es auf der [Homepage](#) in einer gratis-Version für sieben Tage zum Testen. Die Vollversion sollte man anschließend auch hier erwerben: Gegenüber der Playstore-Version kommen dann noch einige Features wie Verschlüsselung hinzu, und außerdem sind die Kosten hier geringfügig niedriger...

Dateimanager

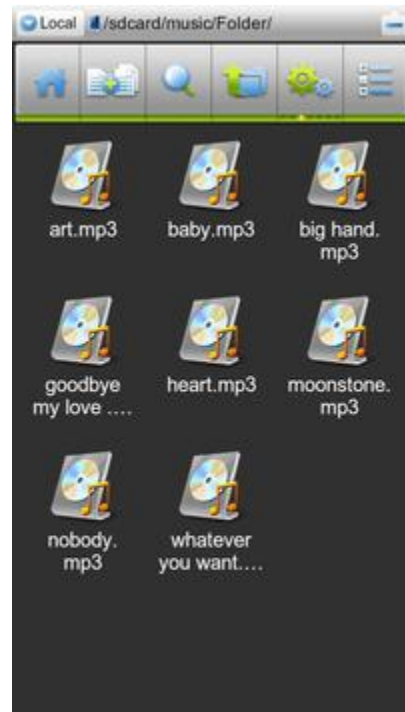
Gibt es ihn, den ultimativen Dateimanager? Viele Anwender werden auf diese Frage ein klares "Ja" zurückgeben. Nur der Rest der Antwort – nämlich das nebensächliche Detail, um welche App es sich dabei wohl handeln möge – dürfte dabei sehr unterschiedlich ausfallen. Denn was für den einen "ultimativ" ist, mag für jemand anderen völlig uninteressant sein – das Umgekehrte gilt natürlich ebenso. Doch zum Glück ist auch in diesem Bereich die Auswahl an für Android verfügbaren Apps umfangreich: Angefangen bei solchen, die sich ausschließlich um die SD-Karte kümmern – bis hin zu denen, die nebenbei auch noch den ganzen Rest des Androiden (einschließlich der Hardware) verwalten wollen, ist alles dabei.

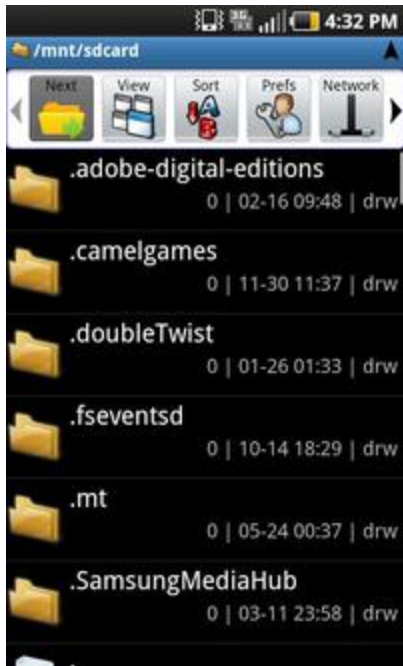
Bei einem derart großen Umfang kann ich natürlich nicht jede App vorstellen. Daher werde ich im Folgenden für die mir wichtig erscheinenden Bereiche die aus meiner Sicht besten Kandidaten herausgreifen – eine Auswahl, die natürlich sehr subjektiv ist. Dennoch hoffe ich, damit einen recht guten Überblick zu geben – und niemandem auf die Füße zu treten. Und gleich vorab: Eine Übersicht zu diesem Thema befindet sich [an diesem Ort](#).

Für den "normalen Anwender"

Mein persönlicher Favorit ist der [ES Datei Explorer](#) (rechtes Bild). Dieser bringt alles mit, was man im Alltag wirklich benötigt: In Listen- oder Icon-Ansicht lässt sich das komplette lokale Dateisystem durchforsten. Bilder, Videos, und auch verschiedene Dokumente kann die App dabei gleich anzeigen, ZIP-Dateien ein- und auspacken, RAR-Dateien auspacken, und mehr. Zugriff auf den heimischen PC aus dem heimischen WLAN heraus? Über [SMB](#) gar kein Problem, sogar den Server findet *ES* selbst. Mal eben etwas vom FTP-Server holen? Ebenfalls kein Thema. Sogar SFTP (SSH), Bluetooth und Dropbox werden unterstützt.

Aber auch an all die, die gern mehr wollen, ist gedacht: Seit einer Weile beinhaltet die App auch Unterstützung für [Wurzelmenschen](#). Ein minimaler Lesezeichen-Manager hilft, wichtige Orte schnell wiederzufinden, und installierte Apps lassen sich auflisten, sichern oder deinstallieren (für Details wird auf das App-Management des Android-Systems zurückgegriffen). Und wem das noch immer nicht reicht, der kann die App mittels weiterer AddOns erweitern: Um einen "großen" Lesezeichen-Manager, einen Task-Manager, einen Sicherheits-Manager...





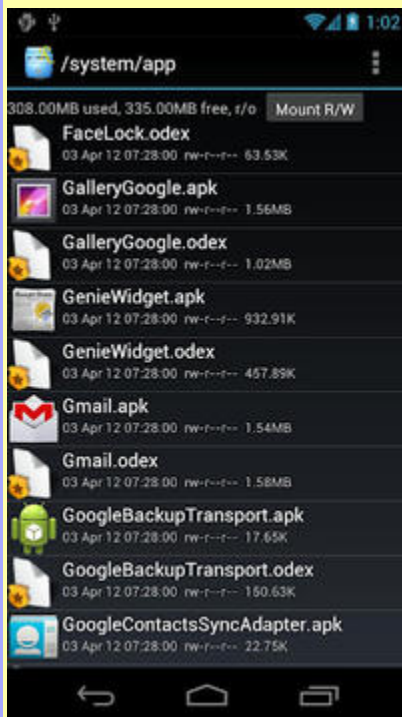
Eine der wenigen Sachen, die ich beim *ES Datei Explorer* erst lange suchen musste, ist die Anzeige von Details zu den Dateien in der Listenansicht. Dies scheint für den **ASTRO Datei-Manager** (linkes Bild) selbstverständlich zu sein. Davon abgesehen scheint sich der Funktionsumfang dieser beiden Apps annähernd zu decken: Auch *ASTRO* versteht sich auf die Navigation durch das komplette Dateisystem, einschließlich dem Kopieren, Verschieben, oder Löschen von Dateien. Mittels Modulen lässt er sich erweitern, so dass gleiches auch für Bluetooth, SMB und SFTP gilt. Ein Task-Manager scheint bereits an Bord, und für das Betrachten von Bildern sowie die Verwaltung installierter Apps ist ebenfalls gesorgt.

"Wurzeltools" bringt *ASTRO* keine mit, das Design ist sicher Geschmacksfrage. Auf der Beliebtheits-Skala liegen die beiden genannten Apps relativ dicht beisammen, was die Bewertungen im Playstore betrifft.

Doch die Szene könnte durchaus in absehbarer Zeit ein wenig aufgemischt werden – denn einer der bekanntesten Dateimanager aus der Windows-Welt hat den *Playstore* betreten. Die Rede ist vom **Total Commander** – und der kann sich offensichtlich in Sachen Funktionsumfang mit den beiden anderen vorgestellten Kandidaten durchaus messen. Wie der Screenshot zur Rechten erkennen lässt, kann man durch das lokale Dateisystem ebenso navigieren wie durch SMB-Freigaben. Ein Lesezeichen- sowie ein App-Manager sind zu erkennen, ebenso die Möglichkeit zum Download weiterer Plugins. Die Archiv-Unterstützung (ZIP, RAR) entspricht der des *ES Datei Explorers*; an Netzwerk-Protokollen stehen neben SMB offensichtlich noch FTP und FTPS zur Verfügung (SFTP ist unter Windows eben nicht so verbreitet). Ebenso soll ein Text-Editor direkt integriert sein. 4,9 Sterne bei über 6.000 Bewertungen (vor allem in so kurzer Zeit) legen nahe, dass der *Total Commander* sicher keine schlechte Wahl darstellt.



Spezielles für "Wurzelmenschen" (root)



Wie bereits gezeigt, lässt sich der *ES Datei Explorer* auch als "root-Explorer" nutzen – allerdings noch nicht sehr lange. Auch der **Ghost Commander** ist dazu in der Lage. Dennoch gibt es natürlich Apps, die sich genau auf dieses Gebiet spezialisiert haben.

Eine dieser Apps trägt den Namen **Root Explorer**, und ist auf dem linken Screenshot zu sehen. Selbstredend lässt sich damit das gesamte Dateisystem erkunden – und was nicht schon beschreibbar ist, wird beschreibbar gemacht (wie man dem Button in der rechten oberen Ecke richtig unterstellen darf).

Doch damit ist das Thema noch lange nicht erschöpft: Mit dabei ist auch ein SQLite Database Viewer, damit man Einblick in die Datenbanken des Systems nehmen kann. Denn davon ist vieles in derartigen Datenbank-Dateien gespeichert – beispielsweise die Anruflisten, die Kurznachrichten, und natürlich auch die Kontaktliste sowie der Kalender. Da sind Dinge wie der integrierte Text

Editor, die Möglichkeit zum Erstellen und Entpacken von ZIP sowie Tar/GZip Dateien bzw. das Auspacken von RAR-Archiven schon fast normal.

Als spezielle **root**-Features kann man da jedoch das Ausführen von Skripten, (Neu-) Einbinden von Dateisystemen, Ändern von Dateiberechtigungen und -eigentümern, sowie den binären APK XML-Viewer betrachten. Lesezeichen und die Möglichkeit, Dateien via Mail, Bluetooth, etc. zu versenden, runden das Ganze schließlich ab.

Auch [SU File Manager & Terminal](#) (rechtes Bild) richtet sich speziell an root-Fans – und erlaubt das Einbinden von Dateisystemen, Anpassen von Dateiberechtigungen, und das Ausführen von Skripten. Ein Terminal (Kommando-Zeilen-Tool) ist ebenfalls integriert. Der Dateimanager gibt Zugriff auf alle lokalen Dateisysteme – aber auch über das Netzwerk auf andere Rechner, wobei wahlweise FTP oder [SMB](#) Verwendung findet.

Ebenso ist es möglich, Textdateien direkt zu editieren, und es lassen sich sogar [symbolische Links](#) anlegen. ZIP-Archive werden direkt unterstützt.



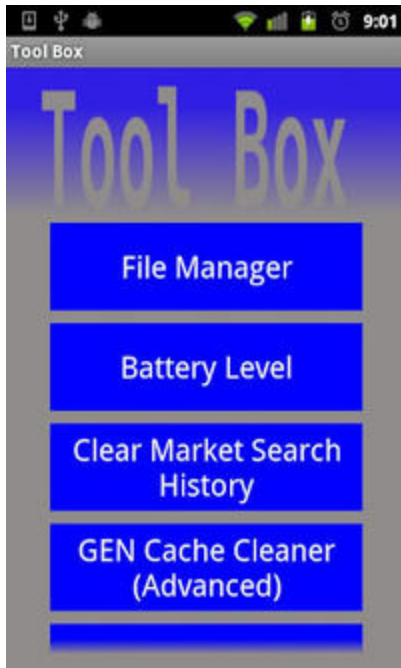
Toolboxen mit integriertem Dateimanager

Um die Sache am anderen Ende zu beginnen: Der bereits [zuvor genannte ES Datei Explorer](#) lässt sich mit AddOns zu einer derartigen Toolbox ausbauen. Da gibt es einen [Lesezeichen-Manager](#), mit dem sich Bookmarks für Dateien, Verzeichnisse, etc. verwalten lassen, einen [Task-Manager](#) (einschließlich eines praktischen Widgets), und sogar einen [Sicherheits-Manager](#). Letzterer vermag es, Apps mit Passwort schützen, hat einen "Thread Detector", kann das Gerät aus der Ferne sperren, und auch den Standort des Gerätes ermitteln.

Von vornherein in Vollausstattung kommt hingegen [eFile](#) (rechtes Bild) daher – und das gratis und ohne Werbung. Einen Großteil des Funktionsumfangs kann man bereits dem Screenshot entnehmen: Dateimanager, Taskmanager, Appmanager, sowie eine Anbindung zu [eRay](#) (einem weiteren Tool des gleichen Entwicklers) lassen sich da erkennen.

Der Dateimanager bietet per FTP und SMB Zugriff auf andere Rechner, kann ZIP-Dateien packen und entpacken, sowie RAR Dateien extrahieren. Suche und Lesezeichen sind mit dabei, [APK-Dateien](#) lassen sich direkt installieren, sowie alle möglichen Dateien per Mail-Attachment oder Bluetooth verschicken. Ein Texteditor findet sich hier ebenso wie die Möglichkeit, das Wallpaper oder den Klingelton zu setzen; sogar Shortcuts lassen sich erstellen.





Prinzipiell interessant klingt auch [Advanced Users Toolbox](#) (linkes Bild): ZIP-Support, App Installer/Uninstaller, Akku-Manager, Market-History-Cleaner, generischer Cache-Cleaner und mehr gesellen sich hier zum integrierten Dateimanager. Wer sein Gerät gerootet hat, kommt in den Genuss weiterer Features – wie etwa einem Cleaner für den Dalvik-Cache.

Noch mehr zu bieten scheint etwa [Advanced Tools](#) (rechtes Bild): An Bord sind u. a. ein Dateimanager mit ZIP-Support sowie Unterstützung für Bluetooth und FTP, ein System-Manager mit umfangreichen System-Infos und der Möglichkeit, etwa

CarrierIQ zu entdecken (falls dieser Spion auf dem Gerät vorhanden ist), App-Manager, Terminal, Sensor-Analyzer, GPS Status & Fix, und mehr. Auch hier gibt es auf gerooteten Geräten erweiterte Funktionalitäten – etwa zur Anpassung der Pixeldichte des Displays, Einfrieren von [Bloatware](#), und mehr.

Weitere Kandidaten dieser Kategorie finden sich noch in der eingangs genannten Übersicht. Alternativ ist es sicher nicht verkehrt, sich für jeden benötigten Bereich eine App auszuwählen, die sich auf diesen spezialisiert hat – und ihre Aufgaben daher mit Bravour löst. Einige davon werde ich im vierten Teil dieses Buches zum Thema [Tuning](#) noch benennen.



Androiden vom PC aus verwalten

Eine Übersicht zu diesem Thema findet sich [an dieser Stelle bei AndroidPIT](#).

Für den "normalen Anwender"

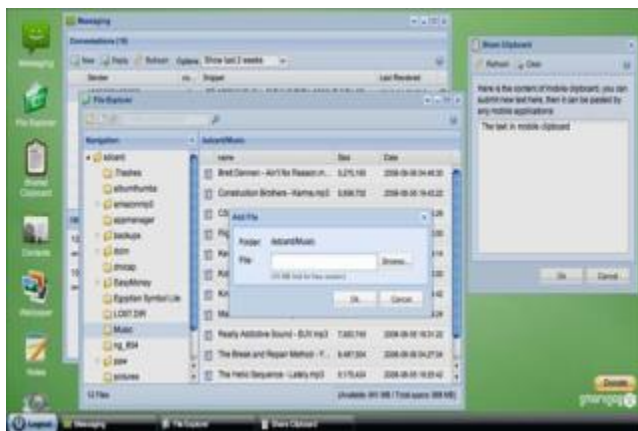
MyPhoneExplorer (rechtes Bild) ist in diesem Bereich sicher die bekannteste und beliebteste App. Ihre größte Einschränkung ist jedoch, dass sie nur mit einem Windows-Betriebssystem kommunizieren kann, auf dem überdies dafür auch eine spezielle Anwendung installiert sein muss. Mac und Linux-Nutzer stehen also außen vor. In dieser Richtung ist auch nichts geplant; es wird jedoch darauf verwiesen, dass die Software mit einigen Einschränkungen auch unter Wine eingesetzt werden kann.



Die meisten Leser wird diese Einschränkung nicht weiter stören, da sie auf ihren PCs das entsprechende Microsoft-Betriebssystem im Einsatz haben. Und für diese ist *MyPhoneExplorer* sicher eine gute Wahl. Sehr bequem in der Handhabung, deckt er vieles ab: Verwaltung der Kontakte, Telefonlisten, Kalender und auch SMS sind möglich, wobei sich auch Anrufe initiieren und SMS lesen/schreiben lassen. Dateien und Medien (Fotos, Videos) lassen sich natürlich ebenso verwalten.

Neben der "Verwaltung" ist sicher noch ein weiterer Bereich interessant: Der Datenabgleich, auch Synchronisation genannt. Und in diesem Umfeld bietet *MyPhoneExplorer* weitreichende Unterstützung: Ein Abgleich ist mit Outlook, Thunderbird, Sunbird, Lotus Notes, Tobit David, Windows Kontakte, Windows Kalender, und weiteren Kandidaten möglich.

Sowohl die App selbst, als auch die zugehörige PC-Anwendung sind kostenlos erhältlich: Erste ganz normal im Playstore, und letztere auf der [Homepage](#) des Anbieters.



Wer sich nicht gern auf ein Desktop-Betriebssystem einschließlich zugehöriger PC-Applikation festlegen lässt, es aber dennoch grafisch nett haben möchte, der sollte einen Blick auf Remote Web Desktop (linkes Bild) werfen. Nomen est omen: Man hat bei dieser App tatsächlich den Eindruck, einen vollständigen Desktop vor sich zu haben. Und das ganz einfach im Web-Browser, ohne Bedarf an zusätzlicher

Software! Auch ein Datenkabel ist recht überflüssig, da alles über WLAN ablaufen kann.

Die Dateiverwaltung ist wahlweise über den integrierten Dateimanager (im Browser) – oder aber auch mit beliebigem Client unter Nutzung des ebenfalls integrierten FTP-Servers möglich. Noch eins oben drauf gesetzt: Auch ein **VNC** Server und Client sind mit an Bord. Ein WiFi-Keyboard registriert sich auf dem Androiden als Eingabe-Methode, und lässt sich so als alternative Tastatur verwenden.

Des weiteren bietet die App die Möglichkeit, vom PC aus Kurznachrichten zu lesen/schreiben, die Kontakte zu verwalten, Screenshots zu erstellen, die WebCam zu nutzen, Apps zu sichern, und vieles mehr. Sogar ein persönlicher Webserver lässt sich damit auf dem Androiden realisieren – oder der Zugriff für einen irgendwo in der Ferne sitzenden Spezialisten via Netzwerk-Brücke umsetzen, während man gerade im mobilen Netz unterwegs ist. Beinahe vergessen hätte ich jetzt die Möglichkeit, dass sich mit Hilfe von *Remote Web Desktop* PC und Androide eine gemeinsame Zwischenablage teilen können...

Auch der **PAW Server** (rechtes Bild) funktioniert unabhängig von einem speziellen PC, und wird einfach im Browser benutzt. Der große Vorteil dieser Tatsache: Man kann ihn überall verwenden – beispielsweise, wenn man bei Freunden/Verwandten zu Besuch ist. Einzig ein funktionierendes WLAN und ein ebenfalls darin eingebuchter Rechner mit Web-Browser werden benötigt. Und da



es wohl kaum einen aktuellen PC ohne Webbrowser drauf gibt, wäre eigentlich ausschließlich WLAN als Voraussetzung zu nennen. Im heimischen Netzwerk eingebucht, und auf dem Router eine entsprechende Port-Freigabe eingerichtet, kann auf diese Weise gar ein Experte aus der Ferne hilfreich unter die Arme greifen – vorausgesetzt, ihm wurden die IP-Adresse und die Zugangsdaten zum PAW Server mitgeteilt.

Wie auch bei den zuvor genannten Kandidaten lassen sich hier Anruflisten, SMS, Kontakte etc. einsehen, Anrufe initialisieren, SMS schreiben, etc. Und wenn der Hund sich "den Knochen" geschnappt und verschleppt hat, selbiger per Knopfdruck zum Klingeln bringen (der "Knochen", nicht der Hund!) um festzustellen, wo er denn nun abgeblieben ist. Vorausgesetzt, der Hund hat dabei nicht das WLAN-Signal verloren (oder war's der Knochen?).

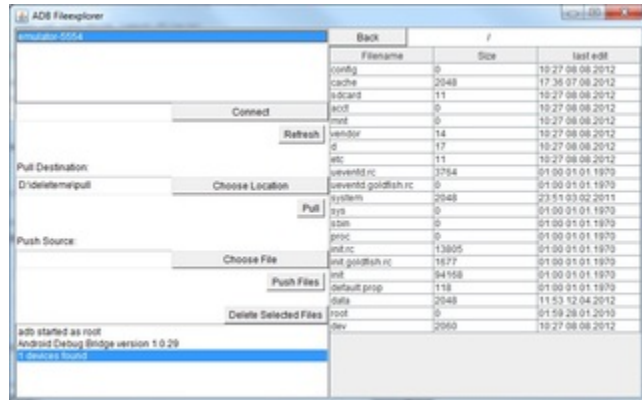
Natürlich ist auch ein Dateimanager enthalten. Fotos lassen sich ebenfalls durchstöbern (auf Wunsch sogar eines davon als neues Hintergrundbild festlegen), der Androide als Diktier- oder Vorlesegerät oder auch Musik-Player oder WebCam nutzen, und vieles mehr.

Für den "Power-User"

Die letzten Absätze müsste ich an dieser Stelle nun wiederholen – denn der *PAW Server* ist definitiv etwas für "Power-User". Besonders interessant für Tüftler: Der integrierte Web-Server kann mit eigenen Skripten ergänzt werden. Dazu bietet sich die *PAW* eigene Skriptsprache an, aber auch ein PHP-Plugin ist verfügbar. Des

weiteren unterstützt PAW auch die im Kapitel [Automatisierung](#) genannten Tools *Locale* und *Tasker*.

Kommen wir zu einem kleinen Schmankerl, welches besonders Android-Entwicklern gefallen dürfte. Sie sind zwar überwiegend im Umgang mit der [ADB](#) gewöhnt, und setzen sicher so manchen Befehl darüber ab – insbesondere, um Dateien mit `adb push` auf das Gerät, bzw. mit `adb pull` vom Gerät herunter zu kopieren. Wer vorwiegend an der Shell unterwegs ist, ist damit sicher durchaus zufrieden – doch so manch einer hat sich dafür vielleicht schon eine kleine grafische Oberfläche gewünscht.



Diesem Wunsch ist XDA Forums-Mitglied [DareTOBe](#) nachgekommen, und hat den *ADB File Explorer* (rechts im Bild) bereitgestellt (siehe [XDA-Developers Forum](#)). In Java geschrieben, lässt sich dieses Programm unter nahezu jedem Betriebssystem einsetzen. Es ermöglicht das Kopieren von Dateien zwischen Gerät und Arbeitsrechner, das Löschen von Dateien sowie das Browsen durch das Dateisystem auf dem Androiden – und auch das Aufbauen einer ADB Verbindung über TCP/IP.

Darf es ein wenig mehr sein? Dann wäre vielleicht [ADBBrowser](#) einen Blick Wert. Verfügbar für Windows und Linux (32-Bit sowie 64-Bit), bietet dieser einen Datei-Browser – aber ebenso auch einen App-Browser. Reicht nicht? Na gut, einen noch: [QtADB](#) (Bild unten). Verfügbar für Windows, Mac und auch Linux (sowohl 32 als auch 64-Bit), ist dieser mit tollen Features nur so vollgestopft: Dateimanager, AppManager, Geräte-Informationen, Verwaltung der SMS, Shell-Zugriff, Screenshots, Logcat, Backup (Nandroid – aber auch einzelne Apps mit und ohne Daten). Und wem es dann wirklich reicht, der findet hier auch eine Möglichkeit zum Reboot...



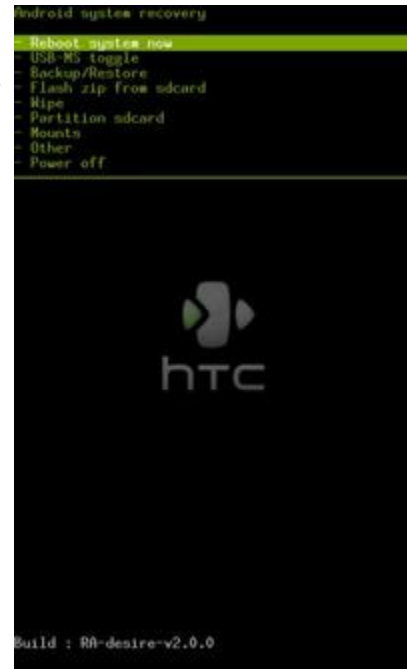
Custom Recovery und ROMs

Die Begriffe [Recovery](#), [ROM](#) und [Custom ROM](#) fielen ja nun bereits des Öfteren. Während die Begriffserklärungen diese Themen nur grob umreißen, soll an dieser Stelle näher auf die Einzelheiten eingegangen werden.

Custom Recovery: ClockworkMod

Was die Begriffserklärung zu [Recovery](#) beschreibt, ist eigentlich schon ein *Custom Recovery* – denn was ab Werk daherkommt, ist eher spartanisch. Nur wenige Punkte stehen hier zur Auswahl – insbesondere Dinge wie [Nandroid Backup](#) oder die Installation von [Custom ROMs](#) fehlen hier gänzlich. Ein Werksreset sowie das Einspielen von vom Hersteller/Netzanbieter offiziell angebotenen Updates ist jedoch meist möglich.

Wesentlich interessanter wird es dann schon mit einem Custom Recovery – welches viele gleich mit dem bekanntesten Kandidaten, dem [ClockworkMod Recovery](#), gleichsetzen. Dies ist einfach das verbreitetste Custom-Recovery, und für die größte Auswahl an Geräten auch verfügbar. Ein anderes recht bekanntes Custom Recovery wäre rechts abgebildetes [AmonRa](#) (welches bei der Brutzelstube [näher beschrieben](#) – aber lediglich für eine Auswahl an HTC-Geräten verfügbar ist). Da zuerst genanntes *ClockworkMod Recovery* jedoch das verbreitetste ist, möchte ich mich hier auf dieses beschränken. Ein Vergleich der Screenshots zeigt außerdem bereits, dass der Funktionsumfang sich stark ähnelt.



Das von [Koushik Dutta](#) (in der Community besser bekannt als *Koush*) entwickelte *ClockworkMod Recovery* (linkes Bild) bietet gegenüber dem *Stock Recovery* eine ganze Reihe zusätzlicher Funktionalitäten, wie beispielsweise:

Erweiterte Recovery, Installations- und Wartungsmöglichkeiten
Backup und Restore ([Nandroid](#))
Installation von Custom ROMs, Kernels, Mods, u. a. m.

[Wipe](#) einzelner/aller Partitionen
Formatieren einzelner Partitionen
Löschen der Akku-Statistiken

Eine komplette Tour durch das ganze Paket einschließlich Anleitungen zur Installation etc. finden sich in einem [Complete Guide](#) (Englisch) bei AddictiveTips.

Besonders praktisch bei *ClockworkMod Recovery* ist, dass der Entwickler auch eine Companion-App namens **ROM Manager** bereitstellt, mit der sich viele Aufgaben auch bequem aus dem laufenden System anstoßen lassen. Wie der Screenshot (rechts) zeigt, lässt sich mit dieser App auch bequem das Recovery-Image installieren. Ebenso findet man hiermit eine Auswahl von Custom-ROMs, die mit dem Gerät kompatibel sind, und kann diese direkt herunterladen und installieren. Ist bereits ein von *ROM Manager* unterstütztes Custom ROM installiert, findet man mit der App auch heraus, ob etwa Updates verfügbar sind – die sich natürlich dann ebenfalls herunterladen und installieren lassen.

Wer des Englischen mächtig genug ist, dem sei auch ein Video bei Youtube nahegelegt: [Detailed Walk-Through of ROM Manager for Android](#) gibt eine Video-Tour durch die Möglichkeiten, die *ROM Manager* bietet.



Wie kann man nun herausfinden, ob *ClockworkMod Recovery* bereits auf dem Gerät installiert ist? Hat man das Gerät gerade neu erworben, erübrigt sich diese Frage sicherlich: Ich habe noch nie gehört, dass ein Hersteller oder Netzbetreiber ein Custom Recovery vorinstalliert hat. Aber sei es drum: Wer ohnehin gern auf den *ROM Manager* zurückgreifen möchte, werfe noch einmal einen Blick auf den Screenshot. Ist nämlich *ClockworkMod Recovery* bereits auf dem Gerät installiert, wird die installierte Version hier auch gleich im ersten Punkt mit ausgewiesen. Alternativ kann man natürlich auch in den [Recovery-Modus](#) booten und schauen, ob die erweiterten Funktionalitäten angeboten werden – und die erste Zeile (wie im obigen Screenshot zu sehen) mit "ClockworkMod" beginnt.

Nandroid Backups

Auch dieser Begriff ist nun bereits häufiger gefallen. Was ein Nandroid-Backup ist, haben die meisten Leser, die bis hierher durchgehalten haben, sicher auch bereits bei der Begriffserklärung "[Nandroid](#)" nachgeschlagen. Doch wie gerade zuvor beim Thema "Custom Recovery", sollen auch hier nähere Details folgen.

Zuerst einmal nehmen wir das Wort auseinander – denn wer hinter dem Begriff "Nandroid" ein mit einem "N" versehenes "Android" vermutet, liegt leicht daneben. Meine Schuld, zugegeben: Ich hätte es deutlicher schreiben können. Dann sähe das Wort nämlich so aus: NANDroid. Und die ersten vier Buchstaben beziehen sich auf das, was gesichert werden soll: Der Inhalt des [NAND-Flash](#), den meisten als "interner Speicher", "Gerätespeicher", oder auch "Telefon-Speicher" bekannt. Wie man selbiges aus dem [Recovery Menü](#) heraus erstellt, wurde bereits im [entsprechenden Abschnitt](#) zum Thema "Backup" behandelt. Von dort ist auch bereits bekannt, dass dabei die kompletten Dateisysteme gesichert werden – ein solches Backup ist also definitiv ein "Komplett-Backup". Ebenfalls bereits an

betreffender Stelle geklärt wurde die Tatsache, dass sich ein Nandroid-Backup normalerweise nur wieder vollständig herstellen lässt (zumindest aus dem Recovery-Menü heraus); wer einzelne Daten benötigt, kann dazu jedoch auf spezielle Apps zurückgreifen.

Verwendet man zur Erstellung von Nandroid Backups gerade behandeltes *ClockworkMod Recovery*, landen die gesicherten Daten im Verzeichnis `clockworkmod/backup/` auf der SD-Karte – wo sie natürlich jede Menge Platz belegen: Schließlich handelt es sich bei jedem Backup um die vollständigen Images der internen Dateisysteme. Daher sollte man hier hin und wieder aufräumen: Während man auch aus Gründen der Datensicherheit eine Kopie dieser Backups auf einem anderen Gerät (z. B. dem heimischen Computer oder [NAS](#)) ablegen sollte, muss man auf der SD-Karte schließlich nicht alle Backup-Generationen aufheben. Löschen kann man ältere Backups auf verschiedene Weise:

- Mit einem [Datei-Manager](#) direkt auf dem Androiden
- Ebenfalls direkt auf dem Android-Gerät mit zuvor genanntem *ROM Manager* (unter "Manage Backups")
- Vom Computer aus, indem die SD-Karte dort eingebunden wird – oder man mit einem [Remote-Manager](#) über WLAN darauf zugreift

In welchem Format die erstellten Backups angelegt werden, hängt vom verwendeten Nandroid-Backup-Tool ab (es handelt sich in der Tat um ein [separates Tool](#), welches lediglich in diverse Custom-Recoveries integriert wurde). So ist etwa *ClockworkMod* mit Version 5 dazu übergegangen, [Tar-Archive](#) zu erstellen. Welches Format man nun genau vor sich hat, lässt sich i. d. R. jedoch leicht am Dateinamen erkennen:

- `system.ext3.tar`: Eine Sicherung der Partition `/system`, deren [Dateisystem](#) EXT3 ist, in einem Tar-Archiv
- `data.img`: Image-Datei der Partition `/data`, wobei das Dateisystem nicht angegeben ist (unter [Dateisysteme](#) wird beschrieben, welche Dateisysteme zum Einsatz kommen)

Und wie kommt man nun am Computer an die Inhalte eines solchen Backups heran – etwa zur Inspektion einzelner Dateien, oder Rettung einzelner Datensätze aus einer der enthaltenen Datenbanken? Bei `.tar` Dateien ist das klar: Auspacken. Aber wie schaut das etwa bei einem YAFFS2-Image aus? Für Linux-Anwender gibt es da das kleine Tool [unyaffs](#), welches das komplette Image entpackt (und zwar in das Verzeichnis, in dem sich auch die Image-Datei befindet).

Auf genannter Website steht sowohl eine statisch kompilierte Binärdatei zur Verfügung – als auch die Möglichkeit, sich den Quellcode zum Selbst-Kompilieren herunterzuladen. Ich habe mir einmal erstere gegriffen: Sie lief problemlos auf meinen 32-Bit Ubuntu 8.04 – und läuft noch immer tadellos auf meinem 64-Bit Ubuntu 12.04 (mit installiertem `ia32-libs` Paket):

- `unyaffs` als root nach `/usr/local/bin` kopieren, damit es im `$PATH` liegt – und somit von überall aus verfügbar ist: `sudo cp unyaffs /usr/local/bin`
- Die zu entpackende Image-Datei in ein leeres Verzeichnis legen (da `unyaffs` scheinbar alles immer dort auspackt, wo diese Datei liegt – und nicht in das Verzeichnis, aus dem es aufgerufen wird): `mkdir mydata && cp data.img mydata/`

- In das Verzeichnis wechseln, und das Westpaket auspacken: `cd mydata && unyaffs data.img`
- Optional: Die Image-Datei wieder löschen (sofern man noch eine Kopie an anderer Stelle liegen hat): `rm -f data.img`
- Auf Forschungsreise gehen

Zur Erforschung der zahlreichen Datenbank-Dateien, die überwiegend im SQLite-Format vorliegen, gibt es zahlreiche Front-Ends: An der Kommandozeile arbeitet etwa das Standard-Tool namens `sqlite3`, als grafisches Frontend für Linux, Mac und Windows gibt es auch [SQLiteMan](#). Und interessante Datenbanken gibt es zur Genüge:

- SMS/MMS: `data/com.android.providers.telephony/databases/mmssms.db`
- MultiMedia Metadaten: `data/com.android.providers.media/databases/*.db`
- Kalenderdaten: `data/com.android.providers.calendar/databases/calendar.db`

Und zahlreiche weitere. Viele Apps speichern ihre Daten auf diese Weise, und sie finden sich zumeist unter `data/<Paketname der App>/databases` wieder (der Paketname lässt sich der Google-Play-URL der App entnehmen – er ist dort mit dem Parameter `id=` angegeben). Fröhliches forschen!

Stock ROMs und AOSP

Irgendwo basiert eigentlich jedes Android-System auf dem [AOSP](#), dem *Android Open Source Project* – den dieses stellt schließlich das Kernsystem zur Verfügung. Nun ist es leider nicht einfach damit getan, sich hier den Code herunterzuladen, zu kompilieren, und auf dem Gerät zu installieren – zu unterschiedlich ist meist die Hardware, sodass Hersteller- und Geräte-spezifische Anpassungen meist nötig sind (außer natürlich für die direkt unterstützten Geräte). Und das ist der erste Punkt, in dem sich sogenannte "Stock ROMs" von den "originalen AOSP ROMs" unterscheiden.

Nebenbei erklärt dies auch viele der Verzögerungen bei Updates: Wenn das AOSP-Team eine neue Android-Version fertiggestellt hat, müssen die Hersteller ja noch ihre spezifischen Hardware-Anpassungen (Treiber etc.) vornehmen. Das ist jedoch nur einer der Gründe – denn schließlich haben sie ja (Dank Open Source) bereits während der Entwicklung Zugriff auf den AOSP-Code, und könnten so auch etwas zeitiger mit ihren Anpassungen beginnen. Da jedoch die meisten Hersteller mehr als nur ein Gerät im Rennen haben, müssen diese Anpassungen auch für mehrere Geräte durchgeführt werden...

Aber es gilt ja nicht nur, die Hardware-spezifischen Dinge anzupassen: Da wären ja auch noch die Hersteller-spezifischen Benutzeroberflächen. HTC Sense, Motorola MotoBlur, Samsungs TouchWiz, und wie sie alle heißen – auch diese müssen mit den "neuen Gegebenheiten" klarkommen. Bis hierher ist das noch einzusehen; doch dann wäre da noch die sogenannte [Bloatware](#), die noch mit rein muss: Apps, die weder vom AOSP, noch von der spezifischen Benutzer-Oberfläche zwingend benötigt werden – von denen man aber meint, dass die Anwender (also wir) nicht ohne leben könnten. Facebook, Twitter, Peep, die ach-so-tolle Aktien-App (die insbesondere für Europäer wenig interessant ist, da sie nur NASDAQ kennt) –

alles Dinge, die sich der Anwender (so er sie denn wirklich braucht) auch bequem über den Playstore installieren kann. Nicht nur, dass viele dieser Apps nerven und Ressourcen fressen – so wirklich loswerden kann man sie wieder nur mit root. Und Apps wie su oder SuperUser.apk sucht man bei der Bläh-Ware vergebens...

Eine weitere Ebene oben drauf bekommen dann noch diejenigen, die ihr Gerät "mit einem Rabatt" bei ihrem Netzbetreiber erworben haben: Dieser möchte natürlich auch noch seinen Senf dazugeben. Oder eher seinen Brandy – nämlich das sogenannte *Branding*. Meist an Dingen wie einer speziellen Boot-Animation mit einem tollen pinkfarbenen "T" zu erkennen. Und an weiterer Bloatware. Das alles muss natürlich bei einer neuen Android-Version auch wieder alles angepasst werden – womit klar ist, warum man hier besonders lange warten darf: AOSP-ROM → Stock-ROM des Herstellers → Stock-ROM des Netzbetreibers. Genau genommen sollte man hier also eigentlich nicht nur vom "Stock-ROM" sprechen, sondern drei verschiedene Typen unterscheiden:

- AOSP-ROM (auch als "plain Vanilla" bezeichnet): Das originale ROM des AOSP. Leider ohne spezielle Anpassungen nicht auf allen Geräten lauffähig
- Stock-ROM: Das vom Hersteller an die Hardware angepasste (und mit einiger Bloatware angereicherte) AOSP-ROM – natürlich passend zum Gerät
- Vendor-ROM: Das vom jeweiligen Netzbetreiber angepasste (und mit weiterer Bloatware angereicherte Stock-ROM – auch passend zum Gerät, nur viiiieel später...

Custom ROMs

Nachdem das Thema der "offiziellen" ROMs geklärt ist, wenden wir uns den Custom ROMs zu. Aus dem Namen ließe sich bereits zweierlei schließen: Dass sie angepasst (aus dem Englischen "to customize" – was nicht direkt "kostümieren" heißt, auch wenn das oftmals ein Nebeneffekt ist), und/oder auf den Kunden (Englisch: "customer") zugeschnitten wurden. Beides ist in gewisser Weise richtig.

Zu unterscheiden sind hier im wesentlichen zwei Gruppen: Die einen basieren auf dem [AOSP](#)-ROM (und pflegen die gerätespezifischen Treiber-Anpassungen selbst nach) – die anderen nehmen eher das Stock-ROM des Herstellers als Grundlage. Bekanntester Vertreter ersterer Gruppe ist sicher [CyanogenMod](#) (das gleich noch separat behandelt wird). Für letztere Gruppe gibt es dann fast so viele ROMs wie Geräte...

Was zeichnet nun ein Custom-ROM aus? Meist kommt es bereits in gerooteter Form daher. Und oftmals läuft es stabiler und performanter als jegliches für das gleiche Gerät verfügbare Stock-ROM – da es besser optimiert und an die Hardware angepasst wurde. Doch auch die [Bloatware](#) spielt hier eine Rolle: Die meisten AOSP-basierten ROMs lassen diese nämlich weg, einschließlich der Hersteller-spezifischen grafischen Aufsätze. Dass dann weniger ungenutzte Apps und Dienste ständig im Hintergrund mitlaufen, kommt natürlich der Performance zugute. Stattdessen findet dafür die eine oder andere wirklich nützliche App ihren Weg in das ROM – wie etwa bereits zuvor genannter ROM-Manager.

Bei den Stock-ROM-basierten Kandidaten bleibt hingegen die Bloatware in der Regel drauf, die Hersteller-spezifische Oberfläche ohnehin. Worin dann hier die

Vorteile liegen? In Optimierungen und Anpassungen, sowie hin und wieder auch aktuelleren Android-Versionen. So kann es durchaus passieren, dass ein Hersteller für ein Gerät keine neueren Android-Versionen mehr bereitstellt – für ein anderes, relativ baugleiches jedoch schon. Hier genügen dann oftmals geringfügige Anpassungen, um auch das "alte Eisen" wieder mit "jüngerem Gemüse" zu bestücken.

Und wie steht es um die Update-Geschwindigkeit? AOSP-basierte Custom-ROMs bieten fast ausnahmslos neuere Android-Versionen weit früher an als die Geräte-Hersteller oder gar Netzanbieter (sofern die Geräte überhaupt noch "offizielle Updates" erhalten). Bei den anderen gibt es dann oftmals noch Updates für Geräte, welche die Hersteller schon längst abgeschrieben haben (für die es also gar keine "offiziellen" Updates mehr gibt).

Klingt gar nicht so verkehrt – aber wo bekommt man so ein Custom-ROM nun her? Die meisten finden sich sicher bei den [XDA Developers](#). Wobei "finden" dabei das eigentliche Problem darstellt: Das richtige ROM für ein bestimmtes Gerät aufzuspüren, ist gerade bei einem derart großen und aktiven Forum nicht sonderlich einfach. Wie gut, dass es auch dafür spezielle Anlaufstellen gibt:

- [PDADB](#): Lange Liste (weit über 800 Seiten) für Geräte aller Couleur – allerdings einschließlich Blackberry, Symbian, und so weiter. Keine Filter-Möglichkeit, daher recht unhandlich
- [CommunityRelease](#): Diese Liste lässt sich explizit nach Gerät, gewünschter Android-Version sowie Entwickler filtern
- [TheUnlockr](#): Gruppiert nach Herstellern, und sodann nach Gerät. Scheint sowohl recht aktuell als auch recht vollständig zu sein – m. E. die beste Online-Anlaufstelle!
- Man nutzt einfach bereits genannten *ROM Manager*, der die ihm bekannten mit dem Gerät kompatiblen ROMs anzeigt, auf Wunsch herunterlädt, und auch gleich installiert – sicher die bequemste Variante

100% vollständig ist sicher keine der genannten (oder auch ungenannten) Übersichten – das wäre auch nahezu unmöglich. Doch eine gute Auswahl stellen sie alle dar.

CyanogenMod

[CyanogenMod](#) ist mit Sicherheit das bekannteste Custom-ROM – und der ständig erwähnte [ROM Manager](#) findet natürlich automatisch die aktuellste verfügbare Version auf dem eigenen Gerät (sofern es unterstützt wird). Offiziell unterstützt werden mittlerweile über 80 verschiedene Smartphones und Tablets verschiedener Hersteller. Gehört das eigene Gerät nicht dazu, findet sich oftmals eine passende "inoffizielle Portierung" – die nicht selten irgendwann ihren Weg in die "offizielle Liste" nimmt. Über zwei Millionen Installationen (und der Zähler tickt in rasender Geschwindigkeit weiter) sprechen für sich.



Wie bereits zuvor erwähnt, basiert *CyanogenMod* direkt auf dem AOSP-Code: Hersteller-spezifische Benutzeroberflächen findet man hier also nicht. Ebenso wenig die "tolle" vorinstallierte [Bloatware](#) – obwohl man selbige hier Dank Kollegen

root (der ja gleich frei Haus mitgeliefert wird) recht einfach wieder loswerden würde.

Stattdessen legt das Team um Steve Kondik eher Wert auf sinnvolle Anpassungen und Ergänzungen, die jeder im Alltag benötigt – oder die zumindest keinen durch zusätzlichen Ressourcen-Verbrauch belästigen, der sie nicht benötigt. Um ein paar der (aus meiner Sicht) wesentlichsten Punkte aufzuzählen:

- **Lockscreen-Gesten:** Statt erst den Bildschirm zu entsperren, dann auf den Homescreen zu wechseln, den App-Drawer zu öffnen, zu suchen, etc., definiert man für seine häufig benutzten Apps einfach Gesten – und malt beispielsweise ein "M", um automatisch den Bildschirm zu entsperren und die Mail-App aufzurufen. Oder die Mal-App – je nach Vorliebe.
- **Phone-Goggles:** Schützen u. a. davor, unbeabsichtigt anzurufen, indem sie eine Sicherheits-Abfrage ("Wollen Sie wirklich...?") einblenden. Wem das noch nicht reicht, der kann auch eine Rechenaufgabe zwischenschalten lassen. Das Ganze lässt sich sowohl auf Zeitfenster als auch auf Rufnummern eingrenzen.
- **OpenVPN:** für mehr Sicherheit ([VPN](#) wurde ja bereits im Zusammenhang mit sicherer Übertragung in einem [eigenen Abschnitt](#) behandelt)
- **Incognito Mode:** für "unerkanntes Surfen", welches (möglichst) keine Spuren hinterlässt: Keine History, und Kekse werden bei Verlassen gleich wieder gelöscht
- **DSP Equalizer:** für besseren Sound
- **Vorinstallierte Apps:** wie beispielsweise *Spare Parts* für tiefgreifendere Konfiguration, und weitere – aber nichts, was (ohne ausdrücklichen Anwender-Wunsch) am Akku nuckelt oder das System ausbremst.

Und bevor jemand fragt: Ja, ich weiß, dass CyanogenMod mittlerweile ein neues Maskottchen hat. Mit gefällt aber nun einmal der Andy auf dem Skateboard...

Wem das jetzt noch nicht "customized" genug ist: Da wäre noch ein Ableger zu nennen, der seinerseits wieder auf *CyanogenMod* aufbaut. **AOKP** (Android Open Kang Project) nimmt noch weitere Anpassungen vor, und ermöglicht dem Anwender auch zusätzliche Einstellungen. Allerdings ist hier die Anzahl unterstützter Geräte mit knapp 30 geringer als bei *CyanogenMod*.

NETZWERK

Der dritte Teil des Buches beschäftigt sich intensiver mit Netzwerk-spezifischen Dingen.

Netzwerk-Konfiguration

Ganz allgemein sind einige der hier relevanten Themen ja bereits im zweiten Teil dieses Buches ([Konfiguration](#)) aufgeführt worden – ohne jedoch auf die relevanten Details näher einzugehen. Letzteres soll nun an dieser Stelle nachgeholt werden. Soweit möglich, sollen dabei Bordmittel zum Einsatz kommen. Wo es sich anbietet (etwa, weil es kein Bordmittel gibt, oder sich etwas auf andere Weise viel bequemer erledigen lässt), kommen jedoch auch spezielle Apps zur Sprache.

An und Aus

Nein, die Rede ist hier nicht vom "großen roten Knopf". Und auch nicht von dem speziellen Button, der als "[Shutdown the Internet](#)" bekannt ist. Ich rede hier vielmehr davon, dass man nicht alle Netz-Dienste ständig benötigt – manchmal sind sie sogar explizit unerwünscht, etwa weil sie wie das Roaming im Ausland zusätzliche Kosten verursachen, oder bei gerade nicht verfügbarer Lademöglichkeit zu sehr am Akku nuckeln. Speziell haben daher vier Dinge unsere spezielle Aufmerksamkeit: WLAN, mobiles Netz (ganz allgemein, und Roaming im Speziellen), Bluetooth, und – nicht Lachen – das Telefonnetz.

Wie war das jetzt? Frage an [Radio Eriwan](#): "Warum sollte jemand das Telefonnetz abschalten wollen? Das ist doch bei einem Telefon die zentrale Eigenschaft!" – "Im Prinzip schon, aber..." War da was? Eingehende Telefonate im Ausland? Ein eBook im Flugzeug lesen? Ungestörte Nachtruhe bei Verfügbarkeit des morgendlichen Weckers? Also gut: Es gibt auch für die Deaktivierung der telefonisch-kommunikativen Eigenschaften des Schlaufons offensichtlich hin und wieder vernünftige Gründe. Schauen wir also einmal nach, wo sich bei Verwendung von Bordmitteln die nötigen Schalterchen versteckt haben. Dafür habe ich rechts noch einmal den Screenshot eingebunden, der sich bereits im Kapitel [Drahtlos und Netzwerke](#) findet.

In diesem Menü, welches sich (wie beschrieben) vom androidischen Home-Screen über *Menü* → *Einstellungen* → *Drahtlos & Netzwerke* erreichen lässt, finden wir gleich mehrere Kandidaten. Ganz zu Oberst wäre da der *Flugzeugmodus*. Bei diesem handelt es sich um einen "Sammelschalter", der gleich mehrere Komponenten lahmlegt: Telefon und WLAN im Wesentlichen. Wobei sich das WLAN im Flugzeugmodus meist wieder separat aktivieren lässt – mit dem Punkt unmittelbar darunter beispielsweise. Der Flugzeugmodus versetzt den Androiden in einen sehr stromsparenden Modus: Bei abgeschaltetem Bildschirm wird fast kein Strom mehr verbraucht. Nur ganz abschalten wäre da noch sparsamer. Daher bietet sich dieser Modus für eine ungestörte Nachtruhe an, aus welcher der androidische Wecker den Schlummernden allerdings morgens wieder zurückholen soll.

Ungefähr in der Mitte des Screenshots findet sich dann auch [König Blauzahn](#). Wer kein passendes Headset sein eigen nennt, und auch sonst keine Bluetooth-



Geräte einsetzt, kann den Haken an dieser Stelle gleich permanent entfernen. Ganz am Ende gibt es schließlich den "Knopf" für das mobile Datennetz.

Nicht mehr auf dem Screenshot zu sehen ist ein weiterer Punkt, bei dem es um die Details für das Mobilfunknetz geht. Dort lassen sich u. a. auch Einstellungen für das Daten-Roaming vornehmen: Soll es verwendet werden, oder besser nicht? Die meisten von uns werden es sicher eher abschalten, und nur im Ausnahmefall aktivieren. Bei Geschäftsreisenden mag das naturgemäß etwas anders aussehen. Aktiviertes Datenroaming erkennt man im Ausland übrigens an einem "R" in der Notification-Bar.

Da gerade das Thema "Roaming" fällt: Für die Telefonie ist der Haken, dass es da keinen "Haken" gibt. Dennoch existieren natürlich Möglichkeiten, sich auch um dieses zu kümmern: Entweder indem man im Ausland einfach den Flugzeugmodus aktiviert, und bei Bedarf einfach WLAN anschaltet – oder *vor* der Abreise (also noch zu Hause) die automatische Betreiberwahl deaktiviert, und stattdessen den eigenen Anbieter fest einstellt. Dies geht ebenfalls im zuletzt genannten Menü, unter dem Punkt "Netzbetreiber".

Eine andere Art von "Netzwerk" stellt das [GPS](#) dar, welches bereits unter [Standort und Sicherheit](#) beschrieben wurde – und daher hier lediglich der Vollständigkeit halber noch einmal erwähnt werden soll.

Wer sich nun nicht ständig durch die vielen Menüs der "Bordmittel" hangeln möchte, um "eben einmal schnell" etwas an- oder auszuschalten: Natürlich gibt es dafür bequemere Wege. So sind die wichtigsten Dinge in [Apps mit Schnellzugriffen](#) enthalten – und vor allem gibt es auch [Schnellumschalter](#), sofern es nur um ein "Umschalten" (an/aus) geht...

Datensynchronisation im Hintergrund

Ein Smartphone hat man schließlich nicht zuletzt, um auch unterwegs immer auf dem aktuellen Stand zu sein – viele Apps synchronisieren dafür Daten im Hintergrund. Das ist zwar einerseits recht nützlich, aber auch nicht unbedingt immer erwünscht. Für einige Apps lässt sich diese Synchronisation daher im Menü [Konten & Synchronisierung](#) abschalten (per Default ist sie für jede App angeschaltet). An dieser Stelle verewigen sich vor allem die Google-Apps wie *Google Mail* oder der Kalender, aber auch die Hersteller-Apps wie *HTC Sync* (ebenso deren Nachrichten und Wetter App sowie die dusseligen Aktien, die kaum jemand braucht) oder Motorolas *MotoBlur*.

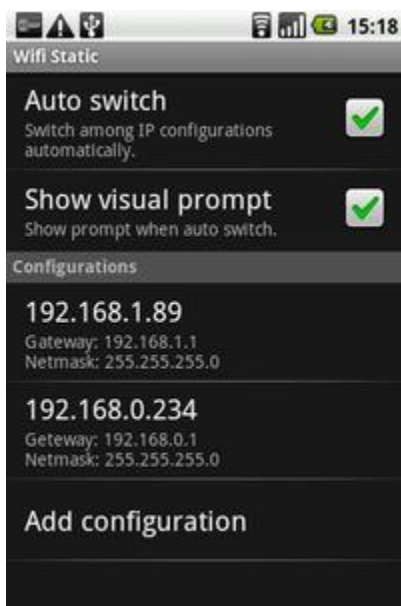
Doch nicht alles, was im Hintergrund Daten austauscht, ist an dieser Stelle auch aufgeführt. Ein klassisches Beispiel sind die meisten RSS-Reader. Wer also Einfluss auf den gesamten "Hintergrund-Verkehr" nehmen möchte, sollte die Einstellungen der verbleibenden Apps separat durchgehen: Bei vielen lässt sich die Häufigkeit der Aktualisierungen einstellen; bei manchen darüber hinaus auch auf ein verfügbares WLAN einschränken.

Mehr zum Thema "Datensynchronisation im Hintergrund" findet sich im Bereich [Tuning](#) unter [Hintergrunddaten und Synchronisierung](#).

WLAN Konfiguration

Besonders viel lässt sich mit den unter [Drahtlos und Netzwerke](#) verfügbaren Menü der Bordmittel nicht einstellen. Allenfalls die "Basics" sind hier verfügbar: WLAN an aktivieren? Bei verfügbaren offene WLANs benachrichtigen? Dazu bei aktiviertem WLAN noch eine Liste der in Reichweite befindlichen Netze mit der Möglichkeit, sich zu verbinden, und allenfalls noch den ggf. notwendigen Netzwerkschlüssel einzugeben. Unter "Netzwerke verwalten" lassen sich auch gerade einmal nicht mehr benötigte, aber konfigurierte Netzwerke (also die, zu denen man bereits einmal verbunden war) entfernen. Mehr ist auf Anhieb nicht ersichtlich. Haben wir da etwas übersehen?

In der Tat. Da war doch am Androiden noch so eine Menü-Taste? Richtig: Dort findet sich ein Punkt "Erweitert". Und was sich dahinter verbirgt, zeigt der Screenshot zur Rechten. In der "Rechtlichen Domain" gibt man an, wie viele Kanäle benutzt werden dürfen – dies unterscheidet sich nach den lokalen Gesetzen, in Deutschland sind es 13 Kanäle. Weiterhin lässt sich der für [Peer-to-Peer](#) zu verwendende Kanal einstellen. Interessant ist ferner die "WLAN Standby-Richtlinie", mit der man festlegt, wann das aktivierte WLAN temporär deaktiviert werden soll. Zur Auswahl stehen hier "bei Display-Aus", "nie wenn im Netzbetrieb", oder "niemals". Wer vermeiden möchte, dass das Internet-Radio im Hintergrund ausgeht, findet hier den richtigen Schalter.



Letztendlich können auch IP-bezogene Einstellungen getroffen werden. Voreingestellt ist hier die Verwendung von [DHCP](#), was in den meisten Fällen auch genau das richtige ist. In manchen Umgebungen ist jedoch eine feste IP notwendig, sowie die damit verbundene manuelle Konfiguration von Dingen wie "Default-Gateway" und "DNS Server" (wer mit diesen Begriffen nichts anfangen kann, braucht das i. d. R. auch nicht). Der Haken an der Sache: Diese Einstellungen gelten global – also unabhängig davon, mit welchem Netz man gerade verbunden ist. Damit sind sie in der Praxis relativ unbrauchbar.

Abhilfe schaffen kann da beispielsweise [Wifi Static](#) (linkes Bild). Mit dieser App lassen sich statische Einstellungen an einzelne Netzwerke binden. Wie der Screenshot erkennen lässt, wechselt die App dann diese Einstellungen auch automatisch, sobald man selbst das Netzwerk wechselt – so sollte das eigentlich von Haus aus sein!

Besonders im "Corporate" Bereich interessant sind darüber hinaus Apps wie der [WiFi Advanced Config Editor](#), mit dem sich WPA Enterprise Parameter wie wpa_supplicant oder auch der Index des WEP-Schlüssels festlegen lassen.

Wer weitere Apps aus diesem Bereich sucht, mag sich vielleicht für [diesen Post](#) im Forum bei AndroidPIT interessieren.

Administration

Genau genommen, gehört die Konfiguration natürlich auch zu den Aufgaben eines Administrators – doch meist wird erst nach ihm geschrien, wenn es irgendwo klemmt. Genau da soll dieses Kapitel ansetzen: Was tun, wenn etwas nicht tut? Welche Hilfsmittel gibt es, um die Ursache(n) eines Netzwerk-Problems zu finden?

Geräte-Informationen abfragen

Zuallererst wären da die Geräte-Informationen, von denen sich einige mit Bordmitteln auslesen lassen – an gefühlten zehn verschiedenen Stellen verteilt: Zunächst unter *Einstellungen* → *Telefoninfo* → *Status* [MAC-Adressen](#), Mobilnetz-Typ und Signalstärke. Dann unter *Einstellungen* → *Drahtlos & Netzwerke* das verbundene WLAN-Netzwerk und ggf. auch IP-Adressen. Weitere Details lassen sich dem Gerät eventuell noch mit diversen [Geheimen Zugriffs-Codes](#) entlocken. Und bevor die letzten Notizen auf dem Zettel angekommen sind, hat der Nachwuchs selbigen bereits zum Malen mit den Buntstiften entführt...

Zum Glück gibt es auch in diesem Bereich wieder eine [ganze Reihe Helferlein](#), von denen ich einige kurz vorstellen möchte. Das Meiste des soeben aufgeführten fasst beispielsweise [qNetInfo](#) (rechtes Bild) zusammen – jedenfalls, soweit es das WLAN betrifft: MAC-Adresse, Signalstärke, verbundenes Netz, zugehörige IP-Adressen, und sogar den "Supplicant Status" (für die Meisten weniger wichtig, aber im "Corporate Context" durchaus relevant). Damit hat sich der Funktionsumfang dieser nicht einmal 20kB umfassenden App aber auch bereits erschöpft.



The screenshot shows the 'qInfo' app interface on a mobile device. At the top, there's a status bar with '3G', signal strength bars, battery level, and the time '9:23 AM'. The app title 'qInfo' is at the top of the screen. Below it, the section 'Wifi Information:' is highlighted. The 'Wifi Information' section contains the following data:

BSSID	
IP Address	0.0.0.0
Link Speed	-1Mbps
MAC Address	
Signal Strength	-200
SSID	
Supplicant State	UNINITIALIZED
Detailed State	IDLE

Below the 'Wifi Information' section, the 'Network Information:' section is visible, containing the following data:

Subnet Mask	0.0.0.0
Default Gateway	0.0.0.0
DHCP Server	0.0.0.0
DNS Server 1	0.0.0.0
DNS Server 2	0.0.0.0
DHCP Lease Time	0

Der erste Punkt lässt sich am Besten mit einem "Zweitgerät" verifizieren: Zeigt dieses verfügbare Netze an, muss die Ursache eine andere sein. Auch das Häkchen auf der Einstellungsseite des Androiden ist schnell geprüft, ebenso der Netzschalter des Routers. Dann bleibt nach dem Ausschlussverfahren noch der letzte Punkt übrig...

...und der Fall, in dem verfügbare Netze angezeigt werden – jedoch der Androide sich nicht zu diesen verbinden möchte. Auch das kann sich auf verschiedene Weise äußern:

- Es sieht zunächst danach aus, als würde die Verbindung aufgebaut – doch plötzlich stehen alle Netze auf "außer Reichweite"
- Eine Fehlermeldung zeigt den fehlgeschlagenen Verbindungsaufbau an

Der erste Fall ist aus dem Leben gegriffen (tritt beispielsweise gern nach Installation des [Custom-ROMs](#) von [CyanogenMod](#) auf dem *HTC Wildfire* auf). In diesem Beispiel liegt das Problem häufig im [Radio-Image](#) begründet; sofern man also seinen Androiden gerootet und mit einer Custom-Firmware versehen hat, hilft meist ein erneutes [Flashen](#) des originalen Radio-Images; doch zuvor erkundigt man sich besser im Forum, da weitere Details entscheidend für den Erfolg sein können.

Der zweite Fall kann mehrere Ursachen haben. Besonders häufig sind:

- falscher Netzwerk-Schlüssel (also so etwas wie ein "falsches Passwort")
- richtiger Netzwerk-Schlüssel, aber beim falschen Netzwerk eingetragen (läuft auf das Gleiche hinaus)
- ein "MAC-Filter" beim Zielnetzwerk ("Betriebsfremden ist der Zutritt verboten!")

Am einfachsten lässt sich der zweite Punkt ausschließen, indem man bei der Auswahl des Zielnetzes im zweiten Anlauf besonders genau aufpasst. Sowohl für den ersten als auch für den dritten Punkt muss man sich an den Admin des Routers wenden: Dieser sollte sowohl den korrekten Schlüssel benennen/mitteilen, als auch ggf. die MAC-Adresse des Androiden in die Liste zugelassener Geräte aufnehmen können.

Minimal anders sieht es aus, wenn es sich um das mobile Datennetz handelt: Hier finden sich die Einstellungen unter *Einstellungen* → *Drahtlos & Netzwerke* → *Mobilnetze* – und man kann keinen falschen Schlüssel beim richtigen Netzwerk (oder umgekehrt) eingeben, da die Verifikation über die SIM-Karte erfolgt. Lässt sich kein Kontakt herstellen, hilft neben einer Überprüfung des Häkchens für die Aktivierung des mobilen Datennetzes noch die Verifikation der [APN-Daten](#). Schafft auch das keine Lösung, bleibt der Anruf beim Anbieter der SIM-Karte übrig...

Server können nicht erreicht werden

Ist ein Server nicht zu erreichen, kann dies mehrere Ursachen haben. Möglich wären unter anderem folgende Probleme:

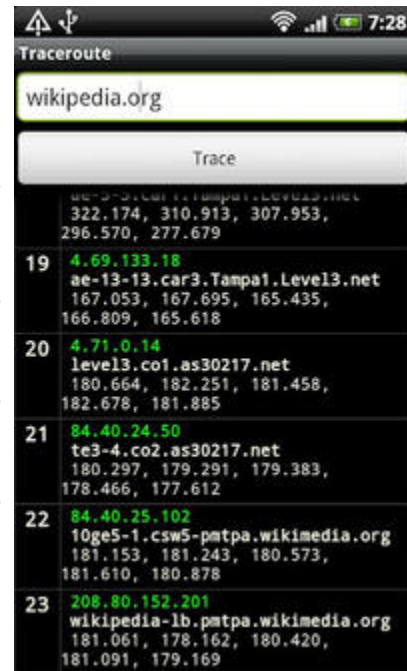
- Der Androide selbst hat keine Netzwerk-Verbindung (siehe [oben](#))
- Der betreffende Server wurde (z. B. für Wartungsarbeiten vorübergehend) heruntergefahren oder gar permanent abgeschaltet
- Der Server an und für sich ist schon erreichbar, nur nicht der gewünschte Service (Beispiel: Beim Zugriff auf eine Webseite kommt der Fehler, der

Server sei nicht verfügbar – Grund ist jedoch nicht ein abgeschalteter Rechner, sondern lediglich die Webserver-Applikation wurde beendet)

- Der Server läuft wahrscheinlich noch – aber das Netz, in dem er steht, ist nicht erreichbar (defekter Router auf der Strecke)

Während sich der erste Punkt noch lokal klären lässt, schaut der Rest zunächst komplizierter aus: Nicht immer kennt man den Betreiber persönlich, um eben einmal kurz anzurufen und nachzufragen. Wie kommt man der Sache aber sonst auf die Spur? Einige passende Tools sind wiederum in [einer Übersicht zusammengefasst](#), eine kleine Auswahl davon kommt hier zur Sprache.

So lässt sich der Weg zum Zielrechner beispielsweise mit [Traceroute](#) (rechtes Bild) nachverfolgen. Endet die angezeigte Liste mit der IP-Adresse des gewünschten Ziels, so ist der Server selbst am Netz – und vermutlich der Service (also Web-App, Mail, oder was immer man benutzen wollte) das Problem. Nicht selten jedoch bleibt der Zug auf der Strecke stehen: Plötzlich dauert es länger, bis die nächste Zeile erscheint, und/oder es kommen nur noch Sternchen. In dem Fall ist das Routing (und wahrscheinlich in irgend einem Rechenzentrum ein Router) defekt; da gibt es nicht viel, was man tun kann. Außer die "defekte Stelle" liegt noch im Bereich des eigenen Netzanbieters: Dann kann man diesen mit den ermittelten Details kontaktieren und um Hilfe bitten.



Traceroute	
wikipedia.org	
Trace	
	322.174, 310.913, 307.953, 296.570, 277.679
19	4.69.133.18 ae-13-13.car3.Tampa1.Level3.net 167.053, 167.695, 165.435, 166.809, 165.618
20	4.71.0.14 level3.co1.as30217.net 180.664, 182.251, 181.458, 182.678, 181.885
21	84.40.24.50 te3-4.co2.as30217.net 180.297, 179.291, 179.383, 178.466, 177.612
22	84.40.25.102 10ge5-1.csw5-pmtpa.wikimedia.org 181.153, 181.243, 180.573, 181.610, 180.878
23	208.80.152.201 wikipedia-lb.pmtpa.wikimedia.org 181.061, 178.162, 180.420, 181.091, 179.169



Zugegeben: Mit derartigen Zahlen- und Zeichenketten ist der Laie nicht selten überfordert. Es fällt einfach schwer, sich darunter etwas vorzustellen. Sicher lässt sich die Ausgabe-Syntax erklären: Die erste Zahl ist einfach ein Zähler (der wievielte Schritt auf dem Weg ist es?), gefolgt von einem Doppelpunkt. Hinter diesem werden IP-Adresse und, sofern verfügbar, der zugehörige Rechnername. Der "große Bruder" auf dem PC hätte zusätzlich noch angezeigt, wie schnell die Reaktionszeit war.

Weitaus anschaulicher stellt [Visual Traceroute](#) (linkes Bild) diese Informationen dar: Darunter kann sich auch ein Laie etwas vorstellen! Die Route wird bei dieser App auf einer Karte angezeigt – allerdings eher grob, indem die Standorte der Stationen mit direkten Linien, die nicht unbedingt der Kabelführung entsprechen, verbunden werden. Dazu gibt es die gleichen, zuvor genannten Details: Im Bild in den "blauen Bällen" zur Rechten die

Schrittnummer, links daneben die zugehörige IP-Adresse und, sofern vorhanden, auch der Rechnername.

Aber Moment, da steht noch mehr: Diese App nutzt [Maxminds GeoIP](#) Datenbank um festzustellen, wo sich die entsprechenden IP-Adressen befinden. So kann sie die Standorte auf der Karte anzeigen, und auch in der Liste benennen. Dazwischen in Grün, und auf dem Screenshot etwas schwer erkennbar die Zeit, die das anfragende Datenpaket für den Weg dorthin und zurück (daher "Roundtrip", was soviel wie "Rundreise" heißt) benötigt hat.

Da diese App allerdings leider aus dem Playstore verschwunden ist (der obige Link führt zu den XDA-Developers, wo man sich die APK-Datei noch herunterladen kann), sei auf die ähnliche App namens [Visual Tracert](#) verwiesen. Diese kann die angezeigten Routen sogar als [KML-Datei](#) für Google Maps und Google Earth exportieren...

Jetzt wissen wir also, dass der Server als solches erreichbar ist. Von einem anderen Rechner aus ließe sich noch feststellen, ob jemand (vielleicht ein Freund, der gerade am PC sitzt) auch auf den gewünschten Service zugreifen kann. Warum können wir das dann nicht?

Antwort auf diese Frage könnte u. a. [MobiPerf](#) (Bild rechts) geben. Neben umfangreichen Informationen zur Konfiguration unseres Androiden und seines Netzwerks kann diese App nämlich auch feststellen, welche Dienste der Netzwerk-Anbieter ggf. gesperrt hat. So zeigt das Beispiel des Screenshots, dass der SMTP-Port 25 nicht genutzt werden kann – hier ist es also auf normale Weise nicht möglich, eine Mail zu verschicken.



anzeigen.

Mit dieser App lassen sich beispielsweise auch die Latenzzeiten für DNS-Lookup/Ping/Verbindungsaufbau, die Bandbreite für Up-/Downloads, Signalstärke und mehr ermitteln. Alle Testergebnisse werden dabei im Cache aufbewahrt. Damit lassen sie sich einerseits auch offline auslesen, andererseits aber ebenso mit späteren/früheren Ergebnissen (oder solchen unterschiedlicher Standorte) vergleichen.

Als recht praktisch dürfte sich auch [Net Status](#) (linkes Bild) erweisen. Bei dieser App handelt es sich um eine kleine Toolbox, in der u. a. folgendes versammelt ist: Ping, Netstat, Arp-Table, und der Scan eines angegebenen Netzwerk-Bereiches zum Auffinden möglicher "Nachbarn". Auch die Netzwerk-Konfiguration des eigenen Gerätes lässt sich



Träges Netz

Sind alle netzwerk-abhängigen Apps durch die Bank grottig langsam, liegt dies sicher weniger an allen zuständigen Servern – sondern vermutlich eher an der eigenen Netzverbindung. Beziehungsweise an der Netzversorgung des Anbieters. Doch wie lässt sich so etwas belegen?

Eine mögliche Antwort ist auf dem rechten Bild zu sehen, und nennt sich [Speedtest.net Mobile](#). Die App eignet sich, um Schwachstellen beim Provider aufzudecken: Die Resultate periodischer Tests werden gespeichert, sodass man im Nachhinein eine Vergleichsmöglichkeit hat. Auf diese Weise ist leicht erkennbar, wo die lahme Ente sitzt – und wo hingegen "Schmitts Katze" abgeht.



cnlab SpeedTest - Resultate

Messungen

2011-02-18	5717 kBit/s	hsi.bluewin.ch
16:34	2792 kBit/s	Indoor
4/8	20 ms	WIFI
2011-02-18	3576 kBit/s	hsi.bluewin.ch
16:34	2644 kBit/s	Indoor
4/7	28 ms	WIFI
2011-02-18	5799 kBit/s	hsi.bluewin.ch
16:33	2731 kBit/s	Indoor
4/6	29 ms	WIFI
2011-02-18	5363 kBit/s	hsi.bluewin.ch
16:19	2688 kBit/s	Indoor
4/5	19 ms	WIFI
2011-02-18	1906 kBit/s	hsi.bluewin.ch
16:15	962 kBit/s	Indoor
4/4	737 ms	Mobile(mixed)
2011-02-18	7204 kBit/s	hsi.bluewin.ch

Zeige 6 Test(s) von 6 total

Statistik

[Zeige Messungen auf der Karte](#)

Übertragene Daten 37 MB (2 MB Mobile)

Wird hingegen ein echter Benchmark-Test gewünscht, greift man beispielsweise zu links abgebildeter App [cnlab SpeedTest](#). Mit dieser lassen sich Messreihen erstellen, die man auch mit Referenzsystemen vergleichen kann. Allerdings erfolgt die Messung hier nicht "nebenbei", sondern es wird separater Datentransfer verursacht: Pro Messung sind dies laut der App-Beschreibung bis zu 7MB.



Wer verbrät mein Datenvolumen?

Wer bereits Android 4.x (4.0 AKA Ice-Cream-Sandwich, 4.1 AKA Jelly Bean) auf seinem Gerät laufen hat, kann diese Frage recht einfach beantworten – denn hier ist die entsprechende Funktionalität bereits von Haus aus im System integriert: Eine Überwachung des Datenverbrauchs bis auf App-Ebene herab (sogar getrennt nach Verbrauch bei Vorder- und Hintergrundaktivität), mit Statistik-Graph, und einschließlich der Möglichkeit zum Festlegen eines Warn- und eines „harten“ Limits (siehe linkes Bild). Natürlich getrennt nach WLAN und mobiler Datenverbindung (siehe rechtes Bild).



Doch auch für den Großteil der Android-Nutzer, die noch nicht in den Genuss dieser Version gekommen sind, sind Hopfen und Malz noch lange nicht verloren – denn für diese gibt es Apps wie beispielsweise [3G Watchdog](#) (linkes Bild). Diese App überwacht die mobile Datenverbindung (3G/Edge/GPRS) und deren Trafficverbrauch, zeigt Benachrichtigungssymbol (Grün, Orange, Rot) in der Statuszeile und gibt eine detaillierte Übersicht zum Verbrauch. Zwei Widgets stehen auch zur Wahl.



Aber *3G Watchdog* kann noch mehr. Sicher: Es warnt vor und bei Erreichen der konfigurierten Limits – was für sich genommen schon eine gute Sache ist. So richtig interessant wird es, wenn außerdem die App [APNroid](#) installiert ist: Kurz vor Erreichen des eingestellten Limits dreht *3G Watchdog* dann nämlich einfach den Hahn zu! Der Zugangspunkt (in der Android-Konfiguration) wird dazu von *APNroid* so verändert, dass er nicht mehr funktioniert. Und bevor jetzt Panik ausbricht: Selbstverständlich lässt sich diese Änderung rückgängig machen...

Ist das zu überwachende Gerät mit einem root-Zugang versehen, lassen sich an dieser Stelle u. U. auch zwei Fliegen mit einer Klappe schlagen: Bereits im Abschnitt [Permissions überwachen](#) wurde die App *LBE Privacy Guard* vorgestellt. Diese kann mehr als nur die Zugriffe auf bestimmte Berechtigungen überwachen – nämlich auch den Datenverbrauch. Nebenbei lässt sie sich auch gleich als Firewall einsetzen, um bestimmten Apps den Netzzugriff generell zu verbieten – oder ihn wahlweise auf WLAN bzw. das mobile Datennetz zu beschränken – wie rechts im Bild ersichtlich.



Remote-Laufwerke mounten

Sicher gibt es eine Reihe guter [Dateimanager](#), mit denen man auf's Netzwerk zugreifen kann. Doch spätestens, wenn es um das Abspielen von Titeln aus einer größeren Video-Sammlung geht, wird das mühsam: Warum erst jede Datei händisch auf den Androiden kopieren? Geht das nicht einfacher?

Die erste Antwort wäre wahrscheinlich "Streaming" – was jedoch in der Regel eine passende Streaming-Server-Software auf der Gegenseite voraussetzt. Viel praktischer wäre es doch, man hätte das entfernte Laufwerk lokal zur Verfügung!

Exakt das versprechen die Apps aus [dieser Übersicht](#) zu leisten. Zum Beispiel der rechts abgebildete [CifsManager](#). Der Name der App deutet es bereits an: Dieser Manager kümmert sich um den [Samba](#) Netzwerkdienst, auch als "CIFS" oder "Windows-Freigabe" bekannt. Einmal konfiguriert, lassen sich so Freigaben auf "Knopfdruck" einbinden und auch wieder entfernen. Und diese Freigaben können sich durchaus auf unterschiedlichen Maschinen befinden – welche selbstverständlich über das Netz erreichbar sein müssen (natürlich über das entsprechende Netzwerk-Protokoll).





Das Ein- und Aushängen der Laufwerke erfolgt hier manuell: Der Nutzer muss sich also selbst darum kümmern. Zwar handelt es sich dabei, nach einmaliger Einrichtung der "Laufwerke", nur um wenige "Tapps" – doch naturgemäß ist der Anwender faul, und möchte selbst das noch vereinfachen. Oder man möchte auf einen Linux/Unix Server per NFS zugreifen.

Die Antwort auf die im Raum stehende Frage heißt **Mount Manager**, ist links abgebildet, und kostet in Vollausstattung ein wenig mehr als zwei Euro. Dafür bringt er aber auch allerhand an Funktionalität mit: Neben Samba/CIFS wird, wie schon erwähnt, auch **NFS** unterstützt. Einen Kernel mit Unterstützung für "loadable modules" und weitere Module vorausgesetzt, wären auch weitere Dateisysteme denkbar. Wenn man die App-Beschreibung, was andere Tools *nicht* tun, so interpretieren darf, dass diese App es "drauf hat", klingt das ganz interessant: Laufwerke werden bei

Einschalten von WLAN aktiviert, sobald man sich im betreffenden Netz befindet. Gleiches gilt für den Start des Gerätes, Tasker-Support, Speicherplatz-Anzeige, und mehr.

Sichere Übertragung

Für harmloses Surfen, ein paar News Lesen, und ähnliche Dinge – da muss man sich über sichere Übertragungswege sicher nicht den Kopf zerbrechen. Anders sieht es aus, wenn sensible Daten ins Spiel kommen: Betriebsgeheimnisse, Finanzdaten, neueste Entwicklungen, oder auch nur die privaten Zugangsdaten zu irgendwas. Beim Gedanken, dass da jemand mitlesen könnte, macht sich beim Anwender ein mulmiges Gefühl breit – während dem Vorstand die Haare zu Berge stehen, als hätte der gerade einen Zehnerpack Koppelzaun in der Hose...

Spätestens bei bildlicher Vorstellung dieses Sachverhaltes wird klar: Dafür muss eine Lösung her. Lässt sich Android tauglich machen für "corporate use"?

Proxy, Tunnel & Co

Manchmal scheint es, als wäre "Gefühl sein alles", und Namen nur "Schall und Rauch" – wie Doktor Faust sich so trefflich auszudrücken wusste (**Faust I, Marthens Garten**). Ist ein **Proxy** nun eine Art spezieller Tunnel – oder nutzt ein **Tunnel** eine spezielle Art von Proxy?

Praktisch gesehen, haben beide viel gemein – und so mancher Anwender weiß die beiden nicht recht auseinander zu halten. Manchem Entwickler geht es offenbar ähnlich. So liest man beispielsweise in der Beschreibung der App **SSH Tunnel**, die sich in erster Linie hinter der ganz speziellen, "chinesische Mauer"

genannten Firewall befindliche Anwender zur Überwindung derselbigen richtet, dass es als "globaler Proxy" fungieren kann...

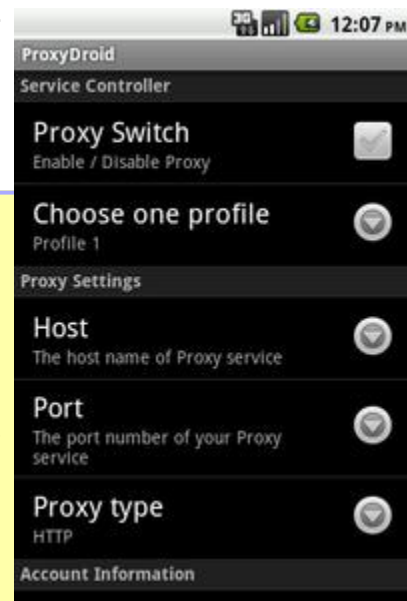
Gemeinsam ist allen beiden auf jeden Fall eines: Sie sind oftmals gleichermaßen geeignet, gewisse Sperren zu umgehen. So möchten viele Firmen gern vermeiden, dass ihre Angestellten sich auf dubiosen Webseiten ihre Arbeitszeit vertreiben – und haben daher die für das Web-Surfing i. d. R. zuständigen Port 80 und 443 gesperrt. Der Zugriff muss stattdessen über einen im Haus befindlichen Proxy-Server (der als Vermittler fungiert, und als einziger auf den genannten Ports "rauswählen" darf) erfolgen. Natürlich ist dabei auch Port 22 gesperrt, damit sich niemand auf fremde SSH-Server verbindet. Und auf dem Proxy sind wiederum eine ganze Reihe interessanter Webseiten gesperrt. Meister Schlauberger umgeht dies nun, indem er sich einen Tunnel baut: Daheim lauscht ein SSH-Server auf Port 443, und dahinter ist der Weg frei. Da Port 443 vom Firmen-Proxy für sichere (und verschlüsselte) Web-Verbindungen akzeptiert wird, lässt dieser unseren Meister Schlauberger brav durch...

Und wie geht man das Ganze nun unter Android an?

Proxies

Grundlegende Proxy-Einstellungen sind unter Android vorhanden – nur nicht immer für den Anwender zugänglich. Fehlt der entsprechende Menüpunkt, lässt er sich mit [HTTP Proxy-Settings](#) dennoch öffnen.

Wem diese Grundfunktionalität nicht ausreicht, der greift beispielsweise zu [ProxyDroid](#) (rechtes Bild). Diese App will helfen, mit allen Apps auf dem Androiden auch über Proxies auf das Internet zuzugreifen (benötigt dazu allerdings [root](#)). Dies ist z. B. häufig in Firmen nötig, wo eine Firewall den direkten Zugriff unterbindet. *ProxyDroid* unterstützt dabei HTTP / SOCKS4 / SOCKS5 Proxies, basic / NTLM / NTLMv2 Authentication, verschiedene Profile und mehr. Es lässt sich auswählen, welche Apps bei ihrem Zugriff auf das Internet über den Proxy geleitet werden sollen. Auch lässt sich ein konfigurierter Proxy für die Verwendung in einem bestimmten Netz (WLAN SSID / Mobiles Network) einschränken, sowie bei Verfügbarkeit eines bestimmten Netzwerks automatisch aktivieren. Widgets ermöglichen ferner die schnelle (De-)Aktivierung von Proxies, und man kann sich per Klingelton und/oder Vibration sogar über Änderungen des Verbindungsstatus informieren lassen. Damit ist doch eigentlich an alles gedacht, oder?





Wenn das Wörtchen "eigentlich" nicht wäre. Denn zumindest einen weiteren Punkt gäbe es noch: Anonymität. Ein Synonym dafür in der Netzwerk-Landschaft wäre **TOR** - nicht der nordische Donnergott, sondern der anonymisierende Routing-Dienst. Und damit kommt **Orbot** (linkes Bild) ins Spiel, der offizielle Android-Port von **TOR** für Android. Mit dieser App wird der Traffic anonymisiert - der Absender eines Netzwerk-Paketes ist also beim Empfänger nicht mehr erkennbar. Unter Android 2.x funktioniert das laut Beschreibung mit Firefox und dem ProxyMob Add-on, mit **Orweb: Proxy+Privacy Browser** sowie mit Orbot-enabled Apps wie **Gibberbot** (secure chat). Transparenter Proxy-Support soll auf den meisten gerooteten Geräten verfügbar sein. Unter Zuhilfenahme anderer Apps (wie dem bereits genannten *HTTP Proxy Settings*) soll es wohl auch mit anderen Apps funktionieren.

Allerdings könnte dieser Zusatzaufwand mittlerweile obsolet sein: Ein aktueller Screenshot (siehe links) legt nahe, dass *Orbot* sich mittlerweile selbsttätig sämtlichen anderen Anwendung zur Verfügung stellen kann.

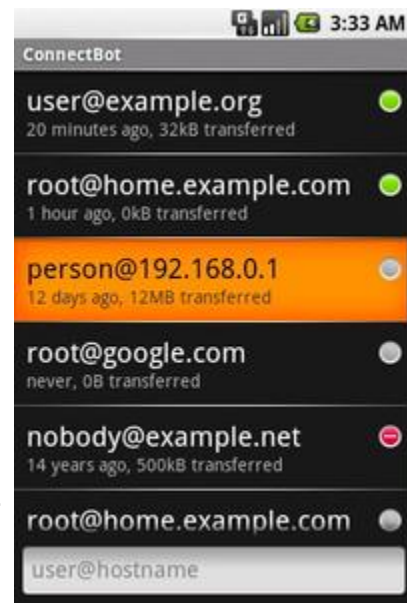
Tunnel

Alles zappenduster? Oder was für ein Tunnel? Manchmal gibt es halt keinen direkten Weg, und man muss Zwischenstation machen. Von A nach C geht es dann nur über B. Und das nennt sich **Tunnel**: "über B nach C". Stark vereinfacht ausgedrückt, natürlich - denn häufig kommt hinzu, dass man dazu in einen speziellen "Schutzanzug" muss. In der Regel hat das auch etwas mit Datensicherheit zu tun. und meistens auch mit SSH...

Als erste Wahl zu diesem Thema wäre sicherlich **ConnectBot** zu nennen. Bei der App handelt es sich um eine einfache, mächtige, open-source SSH-Client-Anwendung - die es ermöglicht, mehrere simultane SSH-Sessions zu betreiben, sichere Tunnel aufzubauen (ah!), und mehr - sogar Copy und Paste zwischen Anwendungen wird unterstützt. So kann man auch "mal eben kurz" auf die Kommandozeile des Linux/Unix Servers zugreifen, der am anderen Ende der Welt steht...

Da sich die Einrichtung eines sicheren Tunnels ohne Vorkenntnisse nicht so ganz trivial gestaltet, und auch ganz allgemein die ersten Schritte so manchen Anwender vor das eine oder andere Problem stellen könnten, sei hier noch auf ein paar Artikel im Netz verwiesen:

- [Verwendung von SSH-Keys mit ConnectBot](#)
- [SSH-Tunnel mit ConnectBot und ProxyDroid](#)



- [ConnectBot Basic Features](#) (Youtube Video)

Weitere Kandidaten (sowohl für den Tunnel als auch zum Thema Proxy) finden sich in [dieser Übersicht](#) bei AndroidPIT.

VPN

Ein **VPN** ist eigentlich so etwas wie ein "systemweiter Tunnel", mit dem der Client (in unserem Fall der Androide) sich in das Netzwerk des Servers einbindet, als wäre er direkt dort vor Ort. Ich bin also zu Hause, und mein Androide ist im Office der Firma – obwohl ich ihn in der Hand halte. Und nein, meine Arme sind nicht mehrere Kilometer lang (ich kann auf Anfrage aber gern noch einmal nachmessen).

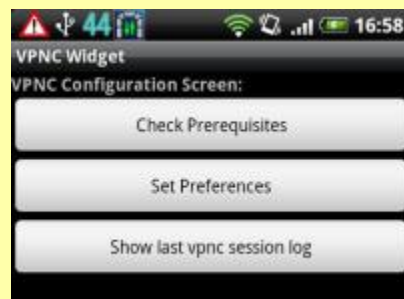
Eine Angelegenheit, die besonders im Unternehmens-Umfeld (und dort speziell für Außendienst-Mitarbeiter) interessant ist, wo man gewisse Dienste nur im **IntraNet** bereit hält. Was sich aber gut auf den Privatbereich übertragen lässt: Auch im heimischen Netz gibt es einiges, was man nicht unbedingt der Öffentlichkeit zur Verfügung stellen, aber dennoch selbst von unterwegs verfügbar haben möchte. Etwa das Web-Interface des eigenen Routers, die Drucker, oder die P...hotosammlung. Welche Helferlein uns zum Thema VPN zur Verfügung stehen, zeigt [diese Übersicht](#) bei AndroidPIT auf.

Android bringt bereits von Haus aus einen VPN-Client mit, der auch mit den meisten VPN-Servern kompatibel ist. Trifft letzteres nicht zu, sind oft tiefer greifende Maßnahmen nötig – da die nötigen Tools der Drittanbieter in der Regel **root** voraussetzen. Aus diesem Grund wird der vorinstallierte Client als "erste Wahl" angesehen: Ein super sicherer Spezial-VPN-Server erhöht die Sicherheit schließlich nicht, wenn im Gegenzug sämtliche mobilen Teilnehmer ihre Geräte rooten müssen...

Die Einrichtung eines VPN mit Android-Bordmitteln beschreibt ein [sechs-seitiger Artikel bei TECChannel.DE](#) im Detail. Wie in bereits genannter Übersicht, wird auch dort auf die Verwendung von [VPN Show](#) hingewiesen, welches einen Schnellaufgriff auf die im Gerät vorhandenen VPN-Einstellungen bereitstellt.

Server von z. B. SonicWALL oder Cisco (Support für Cisco wurde erst [mit Android 4.0 AKA Ice Cream Sandwich integriert](#)) verhalten sich jedoch nicht immer kompatibel zu den VPN-Standards PPTP bzw. L2TP. Stattdessen stellen deren Anbieter eigene Android-Apps zur Einrichtung der VPN-Verbindung zur Verfügung, welche jedoch **root** voraussetzen.

Ähnlich sieht es auch bei der Fritz!Box aus. Die leicht peinliche [Aussage von AVM](#) zum Thema lautet: *Für Apple iOS-Geräte ist die Nutzung per VPN aber grundsätzlich möglich. Eine VPN-Verbindung von Google Android zur FRITZ!Box wird hingegen nicht unterstützt.* Apfel ja, Android nein. Was AVM nicht unterstützt, tun dafür andere: Mit dem **VPNC Widget** (rechts) lässt sich ein Zugang schließlich doch realisieren – **root** und ein paar andere "Kleinigkeiten", die in der Beschreibung der App erklärt werden,



vorausgesetzt. Ein "Patchen" der App, wie in einigen Anleitungen beschrieben, ist hingegen nicht mehr nötig.



Eine Alternative scheint für einige Anwender [OpenVPN](#) zu sein – zumal die entsprechende App bei einigen [Custom ROMs](#) bereits vorinstalliert ist. Eine Anleitung für die Konfiguration unter Android findet sich bei [Nodch.DE](#), für den notwendigen Server ist natürlich ebenfalls Sorge zu tragen.

Eine echte Alternative für viele Anwender hingegen könnte [NeoRouter](#) (linkes Bild) darstellen: All die lästigen Voraussetzungen wie *root*, separates Kernel-Modul und aufwändige Installation scheinen bei dieser Lösung zu entfallen. Auf der [Homepage](#) werden Software-Pakete für alle möglichen Systeme (darunter Linux, Mac, BSD, Windows) angeboten, damit auch die Gegenstelle versorgt ist. Das Ganze verspricht eine einfache Umsetzung ("einfacher als OpenVPN und schneller als LogMeln Ignition"). Für den Einsatz im Unternehmensumfeld steht eine

stabile, für Version-Junkies eine häufig aktualisierte Version zur Verfügung. Beide sind gratis.

Zum Thema "Unterstützung für Fritz!Boxen" hat sich Neo allerdings nicht geäußert. Im [NeoRouter Forum](#) findet sich diesbezüglich lediglich eine unbeantwortete Frage vom Oktober 2010...

AdBlock

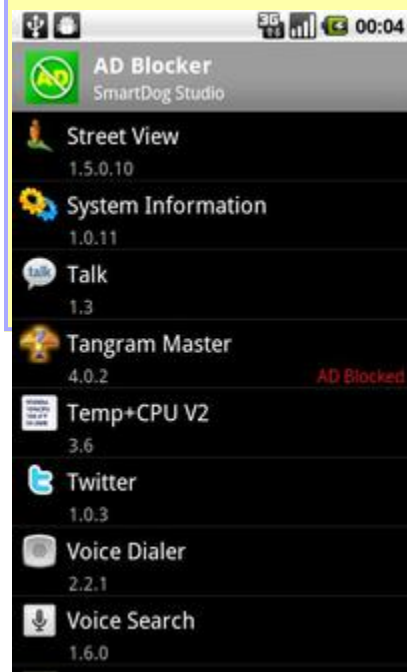
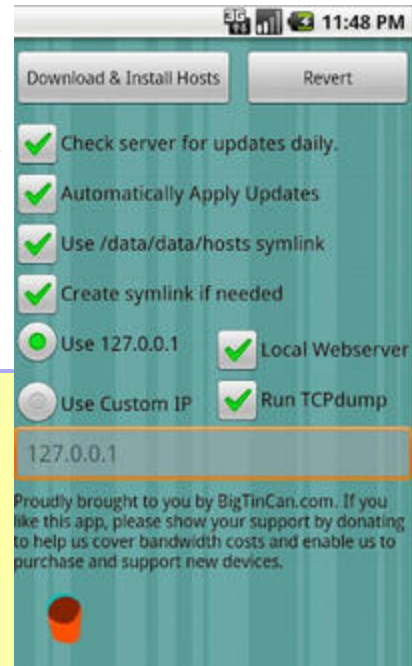
Oh, ich höre schon das Geschrei: Die armen Programmierer, die am Hungertuch nagen müssen... Vielleicht sollte ich in meine Bücher auch ein wenig Werbung einbauen 🤔

Damit mich keiner falsch versteht: Ich möchte niemandem die Butter vom Brot nehmen (die Wurst gleich gar nicht). Eben so wenig soll dies ein Aufruf zur kompletten Reklame-Blockierung sein. Aber es gibt hin und wieder durchaus berechtigte Gründe, warum ein solcher "Ad-Blocker" ans Werk gehen muss. Beispiele gefällig? Oh, "fällig" ist da richtig gut: Abo-Fallen. Erwachsene sollten Dank vollständig aktualisiertem und mit allen nötigen Zusatz-Modulen versehenem [GMV](#) halbwegs darauf achten können, wo sie hintippen. Von einem nicht vollständig ausgewachsenen Wesen (auch "Kind" genannt), dem der Androide "eben einmal für ein Spiel" überlassen wird, kann man dies hingegen nicht unbedingt erwarten.

Dann wären da ja auch noch die ganzen Web-Bugs und andere Tracker, die gern das Nutzer-Verhalten protokollieren wollen. Natürlich nur zu unserem Besten, versteht sich (gemeint ist damit jedoch, dass sie uns "zum Besten halten" wollen). Spätestens hier fallen natürlich wieder die Worte "Datenschutz" und "Privatsphäre", und damit wird es Ernst.

Zum Glück gibt es einige Möglichkeiten, sich zu wehren. Diese benötigen jedoch fast ausnahmslos root.

Am bekanntesten ist sicherlich das rechts abgebildete AdFree Android. Diese App ersetzt die Hosts-Datei mit einer Version, in der die meisten bekannten Ad-Server ins Daten-Nirvana geschickt werden, indem ihnen eine "leere" IP-Adresse zugewiesen wird. Dies blockt Werbe-Zugriffe effektiv – nicht nur im Browser, sondern in allen Apps.



(Hinweis: Zur Wiederherstellung des Ursprungs-Zustandes nicht einfach die App deinstallieren, sondern vorher mit Hilfe der App die ursprüngliche Host-Datei wieder herstellen ("Revert"). Adfree macht ja nichts weiter, als diese Datei einmalig (oder einmal täglich) anzulegen – die Filterung der entsprechenden Netzzugriffe erledigt das Android-System anschließend selbst!)

Ohne root kommt hingegen AD Blocker & Net Toggle (linkes Bild) aus. Mit einem ganz einfachen Trick: Für die ausgewählten Apps gilt einfach die Regel, dass bei ihrem Start das Netzwerk deaktiviert wird. 100% effektiv. Auf Webseiten wird dabei dummerweise nicht nur die Werbung geblockt (was natürlich auch für andere Apps gilt, die außer Werbung noch andere Dinge im Internet zu tun haben). Kleiner Gag nebenbei: Ein Werbe-Blocker, der Werbung anzeigt? Man betrachte den Screenshot einmal ein wenig genauer. Doch was dort wie eine Werbe-Einblendung aussieht, soll wohl nur der Namenszug der App selbst sein...

Geht es lediglich um die Werbung im Browser, so gibt es für den einen oder anderen davon ein passendes AddOn. Für beliebige Browser hingegen greift [AdBlock](#) (rechtes Bild). Und zwar nach einem bereits [weiter vorn](#) erklärten Prinzip: Die App arbeitet als Proxy – und muss dem System daher als solcher bekannt gemacht werden. Wie letzteres funktioniert, wurde ja bereits beschrieben. Auch diese Variante kommt ohne *root* aus – sollte also für jeden Androidler nutzbar sein.

Eine weitere Variante wurde bereits beim Thema [Firewalls](#) genannt: Mit einer solchen ließe sich der Netzzugang einzelner Apps unterbinden. Eine komplexe Firewall könnte auch für bestimmte Apps lediglich bestimmte Ziele sperren. Dies würde jedoch auch eine komplexe Benutzer-Oberfläche erfordern, weshalb sich wohl noch niemand dieser Aufgabe gewidmet hat. Technisch versierte Anwender greifen daher bei Bedarf zu einer Terminal-App, und verwenden das *iptables* Binary direkt. Das setzt natürlich genaue Kenntnis der Syntax voraus – weshalb man sich zunächst zumindest mit den [Grundlagen](#) beschäftigt haben sollte.



WLAN

Netzwerk und Android – da ist WLAN sicher ein zentrales Thema. Und so soll sich dieses Kapitel einigen WLAN-spezifischen Dingen widmen.

Automatische (De-)Aktivierung

Mit diesem Thema greifen wir dem vierten Teil des Buches ([Tuning](#)) bereits ein wenig vor: Da WLAN bereits im Stand-By Betrieb einen nicht unbedeutenden Stromverbrauch aufweist (etwa das zwanzigfache des GPS-Standby – was aber nur ca. 3% des Verbrauches eines auf niedrigste Stufe eingestellten Displays ausmacht), ist ein Abschalten bei fehlendem Bedarf sicher keine verkehrte Idee. Natürlich ließe sich das manuell erledigen; doch vergesslich, wie wir sind, überlassen wir das besser nicht dem Zufall. Eine Übersicht verfügbarer Werkzeuge für diesen Zweck findet sich natürlich wieder [im Forum](#).

Die einfachste Variante lässt sich mit Bordmitteln regeln, und nennt sich "WLAN Standby Richtlinie". Mittels dieser lässt sich WLAN abschalten, wenn man das Display abschaltet. Oder auch nicht. Oder auch nur dann, wenn kein Ladekabel angeschlossen ist. Für manchen Anwender mag das bereits genügen – doch hat es durchaus auch den einen oder anderen Haken: Etwa, wenn man im Hintergrund Musik von einem Server streamt. Oder periodisch Daten synchronisieren möchte (was dann wohl über das mobile Netzwerk erfolgen würde). Für diese und ähnliche Fälle muss also eine Alternative her.

Die könnte beispielsweise [WiFi Location Toggle](#) heißen, und ist rechts abgebildet. Mit dieser App lässt sich festlegen, an welchen Orten man WLAN benutzen möchte: Betritt man die definierten Zonen, wird WLAN aktiviert – verlässt man sie, wird es wieder abgeschaltet. Wer bereits [Tasker](#) im Einsatz hat, kann auf diese App natürlich verzichten – und setzt das dort mit entsprechenden Profilen um. Zumal *WiFi Location Toggle* auf fünf Standorte begrenzt ist.



Schon ganz clever. Dabei ist natürlich bereits ein größerer Bereich abgedeckt, als vielleicht nötig – denn wer weiß schon so genau, wie weit das WLAN-Signal reicht, welcher Radius dafür definiert werden muss? Diese Überlegungen muss man sich bei Verwendung von [WiFi Auto On](#) (linkes Bild) nicht stellen. Hier werden einfach die gewünschten Netzwerke definiert, und um den Rest kümmert sich die App selbst.

Aber woher wissen diese Apps denn nun, wann man sich im fraglichen Bereich aufhält? Würden sie dafür GPS verwenden, fiel der Stromverbrauch ja noch höher aus. Keine Sorge: Das tun sie nicht. Stattdessen

merken sie sich die in der Nähe befindlichen Mobilfunk-Zellen. Und die spürt ein Mobiltelefon ohnehin auf – werden sie doch für die Telefonie-Funktion benötigt.

Zu Suspekt? Okay, es gibt auch noch andere Möglichkeiten. So fährt beispielsweise **Auto WiFi Toggle** (rechtes Bild) eine komplett andere Strategie. Vereinfacht gesagt, schaltet die App je nach Konfiguration ein aktiviertes WLAN entweder aus, wenn man mit keinem Netz verbunden ist – oder es schaltet WLAN von Zeit zu Zeit an und wartet kurz, ob eine Verbindung aufgebaut wird (andernfalls wird WLAN wieder deaktiviert). Das ist aber nur die einfache Fassung – alles lässt sich bis ins Kleinste konfigurieren. So kann z. B. das Prüfintervall nicht nur global festgelegt werden, sondern für einzelne Zeiträume unterschiedlich. Das macht ja auch Sinn: Wer weiß, dass er generell zwischen Mitternacht und sechs Uhr morgens schläft, braucht in der Zeit ja nicht so oft auf verfügbares WLAN prüfen...



Wer wirklich alle Register ziehen will, kann auch zu **Battery Saver** (linkes Bild)

greifen. Womit wir definitiv in den vierten Teil des Buches eingreifen – denn diese App befasst sich nicht nur mit WLAN, sondern nebenbei auch gleich mit Bluetooth und GPS. Sie kümmert sich u. a. um die Stärke des WLAN-Signals, aber ebenso um das Zellen-Signal und anderes, und reagiert auf konfigurierte Schwellwerte. Wenn ich die Beschreibung richtig verstanden habe, schaltet es die entsprechenden Dinge selbsttätig aus, sobald die Grenzwerte unterschritten werden: Reicht die Signalstärke nicht mehr aus, macht es ja auch keinen Sinn. Zum Beispiel in der U-Bahn einiger Großstädte. Wie viel zusätzliche Akku-Laufzeit sich damit "herauskitzeln" lässt, hängt von verschiedenen Kriterien ab – und ist daher eher ein recht individueller Wert. Auch die Playstore-Kommentare sind da wenig aufschlussreich...



Tethering

Da ist man also unterwegs, und würde gern mal schnell mit dem Tablet oder auch Notebook ins Netz der Netze. Natürlich kein WLAN weit und breit – zumindest keines, dem man traut. Da bietet sich doch förmlich das mobile Datennetz des Androiden-Telefons an, oder? Klar, mit Android 2.2 oder neuer gar kein Thema, da erstellt man sich einfach seinen eigenen Hotspot. Ist ja von Haus aus dabei.

Die zugehörigen Einstellungen finden sich unter *Einstellungen → Drahtlos & Netzwerke → Tethering & mobiler Hotspot*. Je nach Android-Version (und teilweise auch Geräte-Hersteller) heißt dieser Punkt u. U. leicht anders, und bietet auch unterschiedliche Möglichkeiten. So kann eventuell zusätzlich zum WLAN-Tethering auch noch USB-Tethering im Angebot sein.

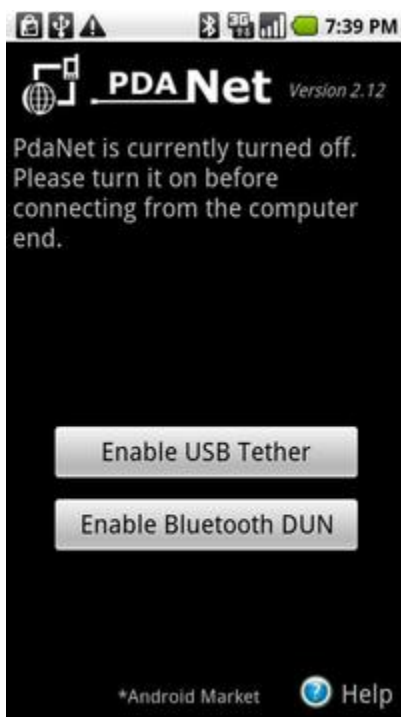
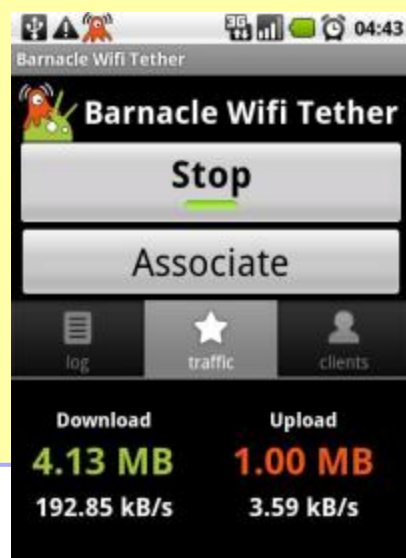
Die Einrichtung ähnelt der eines Routers – was ganz logisch ist, denn schließlich möchte man seinen Androiden ja gerade in einen solchen verwandeln. Dazu müssen zunächst einige wenige Daten festgelegt werden, wie auf dem rechten Screenshot erkennbar: Die zu verwendende SSID für das zu errichtende drahtlose Netzwerk, die Sicherheitsstufe ("offen" und WPA2" stehen dafür i. d. R. zur Auswahl), sowie ggf. ein "Passwort" (auch als "WLAN-Schlüssel" bekannt). Ist das geschehen, lässt sich der "mobile Hotspot" jederzeit aktivieren und deaktivieren.

Doch wie sieht es aus, wenn man eine ältere Android-Version verwendet – oder der Hersteller das Tethering aus dem Menü entfernt hat? Dann erledigt das die App eines Drittanbieters. Vorausgesetzt, man verfügt auf dem Androiden über root. Ohne diesen wird es zwar nicht unmöglich – aber zumindest schwierig und unbequem...



Die wohl beliebteste App in diesem Umfeld ist Wireless Tether for Root Users (linkes Bild). Eine wichtige Voraussetzung zu deren Nutzung ist bereits im Namen enthalten, hinzu käme da noch ein Kernel mit Netfilter-Support. Die meisten Custom-ROMs dürften beide Voraussetzungen erfüllen, während die vom Hersteller installierte Firmware selbst nach rooten häufig den zweiten Punkt nicht erfüllt. Hat man diese Hürden jedoch erfolgreich gemeistert, steht das Internet per WLAN und Bluetooth bereit. In den meisten Fällen als AdHoc (Peer-to-Peer) Netz, auf einigen Geräten auch wie ein gewöhnlicher Hotspot im "Infrastructure" Modus. Als Verschlüsselung kommt 128Bit WEP sowie auf einigen unterstützten Geräten auch WPA/WPA2 zum Einsatz. Als weitere Zugriffskontrolle steht (wie auf der linken Abbildung zu sehen) ein MAC-Filter bereit: So lassen sich unliebsame "Gäste" selbst dann aussperren, wenn sie irgendwie an den Netzwerkschlüssel gekommen sein sollten.

Wer an der genannten zweiten Voraussetzung (dem angepassten [Kernel](#)) scheitert, kann auf [Barnacle Wifi Tether](#) (rechtes Bild) ausweichen. *root* ist allerdings auch bei dieser App nötig, und auf den "Infrastructure"-Modus muss verzichtet werden – was aber in den meisten Fällen locker zu verschmerzen ist. Neben der SSID lässt sich bei *Barnacle* optional auch die BSSID einstellen, zur Verschlüsselung gesellt sich auch hier die Zugriffskontrolle per MAC-Filter. Zahlreiche weitere Konfigurationsmöglichkeiten geben dem Anwender die Möglichkeit, die App an seine Bedürfnisse anzupassen.



Steht *root* ebenfalls nicht zur Verfügung, fällt WLAN als Möglichkeit wohl aus. Es kann jedoch auf [PdaNet](#) (Abbildung links) ausgewichen werden, um das Netz zumindest per USB oder Bluetooth zur Verfügung stellen. Man muss die Dinge positiv sehen: Bei einem Kabel ist klar erkennbar, wer dran hängt – ein "heimliches Einklinken" wird somit deutlich erschwert. Als nette Dreingabe gibt es dazu die Möglichkeit, vom PC aus SMS zu empfangen/senden.

Ist das Tethering einmal eingerichtet, könnte man es auf Knopfdruck aktivieren – wäre der Knopf nicht so versteckt, insbesondere bei den Bordmitteln. Abhilfe schaffen hier etliche Widgets für den Schnellzugriff, die teilweise das Schalten direkt übernehmen. Auch diese sind in der eingangs genannten Übersicht zu finden – zumindest eine Auswahl davon.

Reverse Tether

Was passiert nun, wenn man das ganze umdreht? Klar: Man bekommt den Laptop-Deckel nicht mehr auf. Unsinn: Gemeint ist natürlich eine Umkehr der Verbindungsrichtung. Etwa, wenn am aktuellen Standort nur kabelgebundenes Netzwerk zur Verfügung steht. Dann ließe sich die am Laptop bestehende Internet-Verbindung auch vom Androiden aus nutzen. Per USB-Kabel ermöglicht dies beispielsweise die App [Reverse Tether](#) (rechtes Bild). Sie unterstützt eine automatische Konfiguration, lässt aber auch manuelle Anpassungen zu. Vorausgesetzt, das Android-Gerät wird unterstützt (leider funktioniert die App nicht auf allen Geräten – hier scheint es hin und wieder zu Kompatibilitäts-Problemen zu kommen), steht die Internet-Verbindung pronto zur Verfügung. Alles, was es dazu braucht, ist ein USB-Kabel – sowie ggf. (bei Einsatz von Windows) der entsprechende Geräte-Treiber für den Androiden. Optional kann man sich nun auch bei jedem Anstecken des USB-Kabels automatisch fragen lassen, ob die Internet-Verbindung hergestellt werden soll.



Es gibt eine gratis Testversion, die allerdings hinsichtlich der Verbindungszeit eingeschränkt ist. Damit lässt sich jedoch auf jeden Fall feststellen, ob die App mit dem eigenen Gerät überhaupt funktioniert. Die Vollversion kann man sodann für knapp vier Euro erwerben. Weiteres lässt sich auch (in englischer Sprache) bei [AndroidAuthority](#) nachlesen.

Sollte diese App nicht mit dem Wunschgerät kompatibel sein, gibt es noch ein paar Alternativen. Diese bestehen jedoch nicht aus einer einfachen App, sondern bedürfen oftmals ein wenig Handarbeit (etwa das Anpassen von Systemdateien auf dem Androiden – wie der Datei `/system/bin/wpa_supplicant` zur Ermöglichung des Zugriffs auf Ad-Hoc Netzwerke per WLAN (siehe [XDA-Developers](#)). Oder sie sind auf ein bestimmtes Betriebssystem auf dem PC/Laptop angewiesen – wie im Falle von [Android Reverse Tethering for Windows users](#).

Eine Methode, die sich einfach mit Hilfe einer Terminal-App umsetzen lassen sollte, beschreibt [Rotemmiz](#) in einem [Beitrag bei StackExchange](#):

```
ifconfig wlan0 up
iwconfig mode auto
iwconfig wlan0 essid "your SSID" channel 11 mode auto
ifconfig wlan0 10.0.0.x netmask 255.255.255.0
```

Diese Zeilen gilt es natürlich als root auszuführen werden – wobei "your SSID" durch die SSID des Access-Points, und 10.0.0.x durch die für den Androiden zuständige IP-Adresse ersetzt werden müssen.

Eine weitere Möglichkeit beschreibt [GWLlosa](#), ebenfalls wieder bei [StackExchange](#). In seinem Fall ermöglicht die manuelle Anpassung der Konfigurationsdatei `wpa_supplicant.conf` den Zugriff auf AdHoc-Netzwerke via

WLAN. Glücklicherweise, wer in den erweiterten WLAN-Einstellungen, wie im Screenshot unter [WLAN Konfiguration](#) zu sehen, den Kanal für "Peer-to-Peer Verbindungen" einstellen kann: Ein solches Gerät unterstützt den Ad-Hoc Modus von Haus aus.

Wesentlich einfacher geht es natürlich, sofern der Laptop/PC ein vollwertiges "Infrastruktur" WLAN (im Gegensatz zu "AdHoc") bereitstellen kann, oder man über einen passenden WLAN Reiserouter (bereits für unter 50 Euro im Handel erhältlich) verfügt. Dann nutzt man das bereitgestellte WLAN nämlich einfach, wie jeden anderen Hotspot auch.

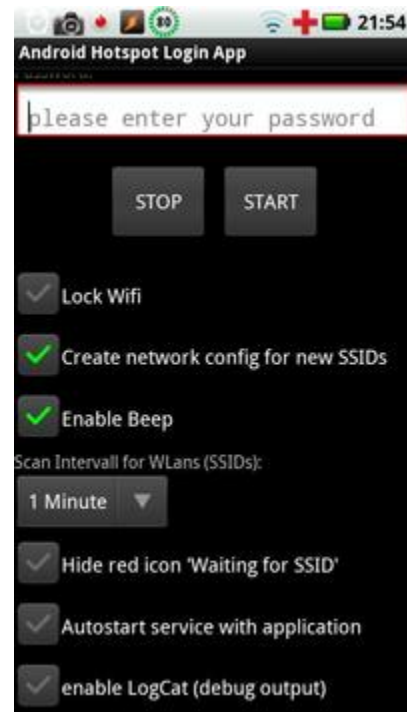
Hotspots & WLAN-Finder

Unterwegs und insbesondere im Ausland, wo man auf teure Roaming-Kosten gern verzichtet, ist so mancher scharf auf ein verfügbares WLAN. Viele Hotels und mittlerweile auch Cafés bieten ihren Gästen auf Anfrage einen Zugang auf diese Weise an. Auch die Telekom hält etliche Hotspots bereit. Die Frage, die sich stellt, ist nur: Wie bzw. wo findet man diese Zugangspunkte?

Natürlich gäbe es da zunächst den in Android integrierten WLAN-Scan, der sich unter *Einstellungen* → *Drahtlos & Netzwerke* → *WLAN-Einstellungen* findet. Während sich dieser wunderbar eignet, so man das Netzwerk kennt, zu dem man sich verbinden will (also etwa das im eigenen Hotel, zu dem der Portier gerade die Zugangsdaten auf einem Kärtchen übergeben hat) – tauchen die ersten Probleme spätestens auf, wenn kein WLAN angezeigt wird. Schon fünfzig Meter weiter könnte eines sein – nur in welcher Richtung? Und ist ein als "offen" gekennzeichnetes WLAN auch wirklich "offen", oder kommt man nach dem Verbindungsaufbau lediglich mit dem Browser auf eine Anmeldeseite?

Eine [Übersicht im Forum](#) nennt eine Reihe von Tools, die genau diese Fragen beantworten wollen. So kann man beispielsweise Telekom-Hotspots relativ leicht aufspüren: Sie lassen sich an ihrer [SSID](#) erkennen. Da die zu verwendenden Zugangsdaten bei allen diesen Hotspots identisch sind, stünde einer automatischen Anmeldung eigentlich nichts im Weg. Zu dumm nur, dass diese jeweils nach dem Aufbau der (unverschlüsselten) WLAN-Verbindung über eine Webseite erfolgt.

[Android Hotspot Login](#) (rechtes Bild) erledigt dies dennoch ohne viel Schnick-Schnack: Einfach die Login-Daten konfigurieren, ggf. das Scan-Intervall anpassen und ein paar Häkchen umsetzen – fertig. Bei Bedarf jetzt den Service noch starten – und sobald ein passender Hotspot in der Nähe ist, verbindet die App automatisch, wobei es ggf. "Piep" macht. So die Theorie. In der Praxis scheinen Änderungen seitens der Telekom die App aus dem Tritt gebracht zu haben. Doch sofern ein nach



September 2011 datiertes Update verfügbar werden sollte, lohnt sich ein Blick auf diese App mit Sicherheit.

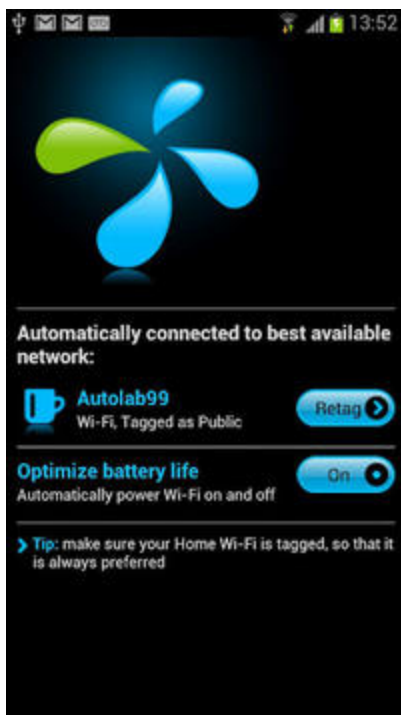
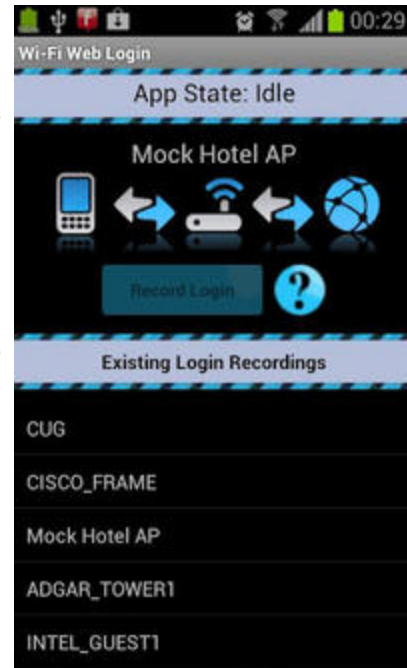


Die "originale App" von der Telekom selbst müsste das Problem doch eigentlich lösen. Werfen wir daher einen Blick auf [HotSpot Login](#) (linkes Bild): Diese App soll unabhängig von der SIM-Karte, oder sogar ohne selbige funktionieren – das wäre doch eine prima Sache für Tablet-Nutzer! Also flugs mal auf mein Testgerät (ohne SIM) gezogen und geschaut. Das Ergebnis ist ernüchternd: Im heimischen WLAN angemeldet, findet die App in München ganze 0 Hotspots, davon 0 in der näheren Umgebung. Wie kann das sein? Beschreibung nochmal lesen: Aha, die Suche funktioniert nur, wenn man an einem Telekom-Hotspot angemeldet ist. Tolle Logik: Um einen Telekom-Hotspot zu finden, muss man in einem solchen angemeldet sein. Dann allerdings braucht man ihn ja eigentlich nicht mehr suchen...

Ich teste ein wenig weiter. Am Hauptbahnhof gibt es ja einen Telekom-Hotspot. Und der wird von der App auch sogleich entdeckt. Die Anmeldung erfolgt *ohne Rückfrage, ob ich das überhaupt will* – und der Text auf dem Screen bestätigt, was wir schon immer ahnten: Bayern gehört nicht zu Deutschland (Text auf dem Screen: "Sie nutzen einen Hotspot außerhalb Deutschlands – es können Roamingkosten entstehen"). Das sorgt natürlich zunächst für einen Lacher – und macht kurz darauf nachdenklich: Roamingkosten? Danke für den Hinweis! Wäre es tatsächlich so, hätte mir die App jetzt Kosten fabriziert, ohne mich zu fragen! Und überhaupt: Warum Roaming-Kosten? Geht es hier nicht um WLAN? Im Prinzip ja – aber die Abrechnung erfolgt ja über die Telekom, und die macht ja Telefon. Folglich: Befindet man sich aus ihrer Sicht im Ausland, fallen Roaming-Kosten an. Kann man verstehen – muss man aber nicht...

Die nächste Überraschung kommt Stunden später. Wieder zu Hause angekommen, verbindet sich das Gerät nicht mehr mit dem heimischen WLAN. Was ist denn jetzt los? In den Einstellungen nachgeschaut: Dankeschön! Die App hat alle gespeicherten WLANs *deaktiviert*! Man darf sich nun also zunächst wieder überall manuell verbinden. Außer beim nächsten Telekom-Hotspot – außerhalb Deutschlands...

Müssen wir auf den entsprechenden Komfort nun also verzichten, und uns jeweils manuell über die entsprechende Webseite anmelden? Zum Glück gibt es noch Alternativen, die sogar weitere Anbieter abdecken können. Wie beispielsweise rechts abgebildetes [WiFi Web Login](#). Auch wenn sich die genannte Webseite wieder einmal ändern sollte, wird diese App dabei nicht unbrauchbar. Man muss sich dann lediglich wieder einmalig manuell anmelden. Denn *WiFi Web Login* schneidet diese manuelle Anmeldung mit (einschließlich etwaiger Eingabefehler), und spielt sie für die automatische Anmeldung wie ein Makro ab. Bei Änderung des Anmeldeformulars muss also lediglich das Makro erneut aufgezeichnet werden. Ein Vorgehen, welches bei jedem derartigen Netzwerk funktionieren sollte. Davon kann man sich drei Tage kostenlos überzeugen – anschließend muss die App für gut einen Euro käuflich erworben werden, damit sie weiterhin ihren Dienst tut.



Damit wäre geklärt, wie man sich automatisch anmelden kann. Um dies tun zu können, muss man jedoch zunächst wissen, wo sich ein passendes WLAN befindet – wobei uns wiederum eine ganze Reihe Apps behilflich sein wollen. Wie etwa das links abgebildete [WeFi](#), welches ich noch aus "Symbian-Zeiten" kenne. Die Entwickler sind also bereits seit längerem in diesem Bereich aktiv, und verfügen über entsprechende Erfahrungen. Entsprechend groß ist auch die dahinter stehende Community.

Auf der [Website](#) kann man bereits vor Urlaubsantritt nachschauen, wo wohl die besten Hotspots am Ferienort zu finden sind. Inklusive Benutzer-Kommentaren natürlich – das macht eine Community aus. Und vor Ort gibt es dann zusätzlich den "On Demand Scanner" – irgendwie müssen neue Spots ja auch in die Liste kommen. Als Mitglied der Community hat man nämlich die Möglichkeit, neu gefundene Hotspots in die Datenbank einzutragen (wie es der linke Screenshot zeigt). Da sich eine recht rege Community an dieser Aktivität beteiligt,

wächst die Datenbank beständig – die Chancen sind also gut, für den Zielort eine Reihe passender Hotspots im Voraus zu finden.

Der On-Demand Scanner zeigt übrigens nicht nur an, dass da "ein WLAN" verfügbar ist. Gefundene Netze werden in mehrere Kategorien unterteilt. Das wären zunächst verschlüsselte (also solche, für die man zunächst einen Netzwerk-Schlüssel benötigt), und offene (ohne Verschlüsselung). Doch letztere werden nochmals weiter unterteilt: *WeFi* erkennt automatisch, ob sich hinter einem vermeintlich "offenen Zugang" noch eine Web-Anmeldung verbirgt. Außerdem werden die Netzwerke besonders hervorgehoben, bei denen eine gute Internet-

Verbindung bereits durch die Community bestätigt wurde. Und das sind nur einige der verfügbaren Details...

Einen anderen Weg beschreitet [WiFi Manager](#), indem er gefundene Netze grafisch darstellt (siehe rechtes Bild). So lässt sich nicht nur einfach erkennen, welches das stärkste WLAN in Reichweite ist – sondern auch, ob es sich nicht eventuell mit einem anderen überlappt, was die Kanäle betrifft (und die Signalstärke damit wieder relativieren würde). Den einzelnen Zugangspunkten lässt sich auch eine Beschreibung hinzufügen, so dass man nicht auf den kryptischen Namen allein angewiesen ist. Alles genau beschrieben findet sich auf der [Projektseite](#).

Durch In-App Payment über Google Checkout lässt sich ein Premium-Package hinzu erwerben, welches u. a. das Wechseln vorhandener Netze per Widget und das Ausschließen unerwünschter Netze beinhaltet.



Eine ganz spezielle Art zu suchen bietet

[SWifis](#) (linkes Bild). Die Hauptfunktionalität besteht natürlich auch hier im Auffinden der Netzwerke und dem Verbinden mit selbigen. Aber gerade bei der Suche hat man hier vielfältige Einstellungs-Möglichkeiten: Mindest-Signalstärke? Oder alle Netze mit einer bestimmten Verschlüsselung? Oder einer bestimmten ESSID bzw. BSSID? Oder nur bestimmte Kanäle? Alles kein Thema mit dieser App, die für ca. einen Euro im Playstore erhältlich ist. Eine gratis Testversion ist leider nicht (mehr) verfügbar.

War-Driving

Kriegsspiele? Panzerschlachten, Schiffe-Versenken? Mit all dem hat *War-Driving* herzlich wenig zu tun. Eher mit "da War doch was...". Schauen wir also zunächst einmal nach, was genau es mit diesem Begriff auf sich hat, und konsultieren dazu die [Wikipedia](#):

Wardriving ist das systematische Suchen nach Wireless Local Area Networks mit Hilfe eines Fahrzeugs. Der Begriff leitet sich von Wardialing ab, einer Methode, durch Durchprobieren vieler Telefonnummern offene Modem-Zugänge zu finden, wobei einige Wardriver die drei Anfangsbuchstaben als Backronym für „Wireless Access Revolution“ sehen (wohl nicht zuletzt, um dem Begriff den martialischen Klang zu nehmen).

Im Prinzip geht es also darum: Wir machen das Gleiche wie auch Google mit seinen tollen Autos, nur in kleinerem Maßstab – und legen eine Karte verfügbarer WLAN-Netze der Umgebung an. Dabei helfen uns Tools wie beispielsweise rechts abgebildetes [Wigle Wifi Wardriving](#). Die App lässt sich sehr detailliert konfigurieren, und somit auf die eigenen Bedürfnisse anpassen. Gesammelte WLANs können, einen Account vorausgesetzt, zur zugehörigen [Community](#) hochgeladen werden, wo sie dann allen angemeldeten Mitgliedern (und nur diesen) zur Verfügung stehen. Die eigene Sammlung lässt sich auch gezielt durchsuchen, z. B. mittels [regulären Ausdrücken](#) über die SSID. Ebenso ist ein Export nach [CSV](#) und auch [KML](#) möglich.



Als weiteren Kandidaten möchte ich noch [G-MoN für Android](#) (linkes Bild) nennen.

Diese deutsche App bezeichnet sich selbst als *WarDriving Scanner & GSM / UMTS Netmonitor und Messtool für Funknetzplaner und Optimierer*. Sie scannt alle in Reichweite befindlichen Netzwerke, und speichert deren Daten einschließlich der GPS-Informationen in einer internen SQLite-Datenbank auf dem Androiden ab. Gefundene Zugangspunkte lassen sich per [KML](#)-Export in *Google Earth* (und sicher auch anderen Anwendungen, welche dieses Format unterstützen) anzeigen, auch ein [CSV](#)-Export ist möglich.

In Reichweite befindliche WLANs zeigt die App auf Wunsch auf in einer integrierten Kartenansicht (siehe Screenshot) an. Wie man dort sieht, werden gefundene Mobilfunkzellen ebenfalls angezeigt; die zu sehende "rote Linie" weist vermutlich vom eigenen Standort zum ausgewählten (und in der Kopfzeile angezeigten) WLAN.

Weitere und ausführlichere Informationen zu dieser App finden sich im [Wiki des Projekts](#) – weitere Apps hingegen in [dieser Übersicht im Forum](#).



Dienste bereitstellen

*Ists möglich, daß so harte Sinnen
Und ungemaine Sprödigkeit
Die zarten Glieder hegen können?
Die mir ein andres prophezeyt.
Hier trifft es nicht von Engeln ein,
daß sie dienstbare Geister seyn.*

(Aus: Johann Christian Günthers, "Als er sich über ihren Eigensinn beschwerte")

In diesem Kapitel wollen wir – abschließend zum Thema Netzwerk – einmal betrachten, welche Dienste ein Androide "nach außen" zur Verfügung stellen kann. Dass ich das Wort "Netzwerk" hier ein wenig weiter fasse, lässt sich bereits an der Überschrift des ersten Abschnittes feststellen:

Kabelgebunden

Wie jetzt: Kabelgebunden? Der Androide hat doch gar keinen Anschluss für ein Netzkabel! Oder vielleicht doch? Im weitesten Sinne kann man ein USB-Kabel ja auch als "Netzwerk-Verbindung" betrachten. Spätestens beim Thema [USB-Tethering](#) wird da jeder zustimmen. Die einfachste Server-Funktion vergisst man jedoch ganz leicht:

Androide als USB-Stick

Mit dem Aufkommen von USB-Anschlüssen und passenden Speichermedien verlor die Floppy-Disk recht schnell ihren Reiz für den "täglichen Datentransport". Dank ihrer einfacheren Handhabung und der größeren Kapazität überflügeln sie auch zügig die CD und DVD – woran auch die bald erschwinglicher werdenden "Rewritables" (wiederbeschreibbaren CDs/DVDs) nichts mehr zu ändern vermochten. Dokumente und mehr wanderten auf USB-Sticks vom Büro nach Hause (und in die Gegenrichtung), Fotos zu Freunden und Verwandten, und so weiter.

Dann kamen die MP3-Player, die nicht viel größer waren als die genannten USB-Sticks – und erfüllten eine Doppelfunktion: Unterwegs konnten sie genutzt werden, um Musik zu hören. Den Datentransport übernahmen sie sozusagen nebenbei.

Heute werden beide gleichermaßen durch Smartphones verdrängt: Mir ist kein Smartphone bekannt, für das es keinen MP3-Player und keinen USB-Anschluss gäbe. Und was spricht dagegen, die eingelegte SD-Karte auch als "externe Festplatte" zu nutzen? Der verfügbare Speicher kann mit dem von USB-Sticks durchaus mithalten, wenn die entsprechende Karte eingelegt ist. Extra Software wird dafür nicht benötigt, und das verwendete Dateisystem ist von Werk aus kompatibel zu allen gängigen Computern.

Eine einfache Sache also – und so mancher greift sich vielleicht jetzt an den Kopf, weil er nicht an diese Möglichkeit gedacht hat. Sein Smartphone hat man ohnehin dabei – warum also noch weitere Dinge einstecken, die man damit bereits abdecken kann? Das Kabel? Auch das ist bei den kurzen Akku-Laufzeiten überall vorhanden. Also hören wir doch auf Altmeister Goethe:

*Willst du immer weiter schweifen?
Sieh, das Gute liegt so nah.
Lerne nur das Glück ergreifen,
Denn das Glück ist immer da.*

(Aus: Johann Wolfgang von Goethe, "Erinnerungen")

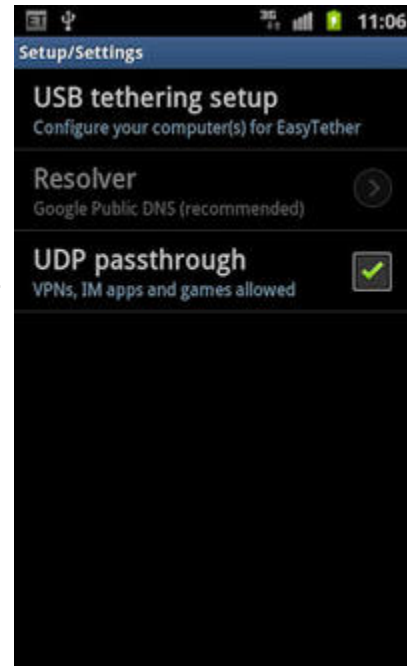
Wem das jedoch nicht "netzwerkig" genug ist: Im [kabellosen Bereich](#) wird unter [Datei-Server](#) beschrieben, wie man den "Datenträger" auch ohne Kabel-Einsatz als solchen vom PC aus nutzen kann.

USB-Tethering

Im Abschnitt [WLAN Tethering](#) habe ich bereits beschrieben, wie man Tethering drahtlos (per WLAN bzw. Bluetooth) bewerkstelligen kann. Welche Gründe könnte es geben, dass man stattdessen lieber ein Kabel nehmen möchte? Da wären tatsächlich ein paar nennenswerte verfügbar:

- Das ans Netz zu bringende Gerät verfügt weder über WLAN, noch über Bluetooth – wohl aber über einen USB-Anschluss (soll selten, aber vorkommen)
- Man möchte potentiellen Lauschern ihr "böses Tun" nicht unnötig erleichtern – in eine drahtgebundene Verbindung können selbige sich wohl kaum einklinken
- Man möchte vermeiden, dass ein zufällig vorbeifahrendes Google-Auto das gerade als Hotspot fungierende Smartphone erfasst und in die Datenbank einträgt
- Die technischen Voraussetzungen für WLAN-Tethering / Hotspot Modus sind nicht gegeben (Android < 2.2, kein root, und keine der verbleibenden Apps funktioniert)

Für den letzten Punkt wurde im genannten Abschnitt ja bereits *PdaNet* genannt, welches ein Tethering per Bluetooth und USB bewerkstelligen kann. Auch das ebenfalls in der zuständigen Übersicht genannte [EasyTether](#) kümmert sich um dieses Problem, und zwar bereits ab Android 1.5 (Cupcake). Es wird dafür kein [root](#) benötigt, und dennoch stehen mit dieser App umfangreiche Features zur Verfügung – wie der Screenshot zur Rechten es bereits erahnen lässt. Nicht umsonst ist diese App so gut bewertet (4,3 Sterne bei ca. 18.000 Bewertungen). Einziger Haken: Die freie Version ist etwas eingeschränkt. Kein HTTPS, und auch "Instant Messengers" werden geblockt. Für die Vollversion werden aber eben einmal knapp acht Euro fällig...





Als Alternative wäre noch [Proxoid](#) zu nennen: Diese App stellt einen Proxy für den Webzugriff von einem über USB-Kabel angeschlossenen Computer bereit, wobei auf letzterem das Android SDK installiert sein muss. Anleitungen für die Installation und Konfiguration auf dem PC finden sich im [Wiki des Projektes](#).

Sind es eher die anderen Punkte, und auf dem "Hotspot-Gerät" läuft bereits Gingerbread (Android 2.3), lässt sich USB-Tethering mit Bordmitteln einrichten (siehe Screenshot links): Der entsprechende Konfigurations-Bildschirm findet sich unter *Einstellungen → Drahtlos & Netzwerk → Tethering & mobiler Hotspot*.

Kabellos

*Setz die Segel, mach die Leinen los,
da draußen warten deine Träume.
Am Horizont ist Gold, siehst du es scheinen?*

(Aus: Peter Maffay, "Der Mensch auf den du wartest")

Also dann: Die Leinen Los! Die Kabel weg! Und den Androiden zum Server umfunktionieren. Wozu das gut sein soll? Da fallen mir auf Anhieb viele tolle Dinge ein: Kabelloser USB-Stick, die Urlaubsfotos ohne Umweg auf einem größeren Bildschirm betrachten... Einige davon sollen im Folgenden vorgestellt werden.

Für all diejenigen, die nicht alles separat lesen – aber alles haben wollen, lohnt sicher ein Blick auf die App [Servers Ultimate](#). Hier scheint wirklich nahezu alles integriert, was in diesem Bereich möglich ist:



Die gratis-Version ist auf maximal zwei gleichzeitig laufende Services beschränkt – was zumindest ausreichend ist, um die Funktionalität zu testen. Die volle Leistung lässt sich sodann mit der Vollversion für ca. 5 Euro ausreizen. Mit jeweils 4,8 Sternen sind die Bewertungen für beide Varianten weit über dem Durchschnitt.

Datei-Server

Das Thema "USB-Stick" hatten wir ja [bereits behandelt](#). Das einzige "störende Element" dabei war das benötigte USB-Kabel. Wollen wir doch Mal schauen, ob es nicht verzichtbar ist! Gleich vorab: Mit Bordmitteln sicher nicht – zumindest habe ich bislang noch nichts entdecken können. Doch wie ich Murphy kenne: Unwahrscheinlich ist es nicht, dass eine neue Android-Version das genau dann anbietet, wenn dieses Buch frisch gedruckt ist...

Beim Thema "Datenserver" denkt sicher so mancher zunächst an [FTP](#). Dies ist auf jeden Fall die ressourcenschonendste Art, Dateien zu übertragen – und so kommt beispielsweise [SwiFTP](#) (rechtes Bild) auch als handliches Paket daher: Gerade einmal 80 kB Download sind es. Einmal installiert und gestartet, stellt die App einen kleinen FTP-Server auf dem Androiden zur Verfügung, auf den man dann von allen Geräten, die den Androiden erreichen können, Zugriff hat. Bei den meisten Systemen genügt dafür bereits ein Webbrowser – in dessen Adresszeile man das eingibt, was einem die App als URL anzeigt. Und schon lassen sich Dateien zumindest herunterladen. Bequemer und definitiv in beide Richtungen geht das natürlich mit einem "richtigen FTP-Client" wie z. B. [FileZilla](#) (sowohl unter Windows, als auch unter Linux und auf dem Mac).



Per Port-Forwarding (oder durch Nutzung eines vom Entwickler bereitgestellten Proxies) lässt sich darüber hinaus auch von beliebigen Orten über das Internet auf diesen Dienst zugreifen. Das gilt bei Nutzung des genannten Proxy sogar, wenn man selbst mit dem Androiden unterwegs und lediglich in das "mobile Netzwerk" eingebucht ist.

Leider ist diese App aus dem Playstore verschwunden, und der Entwickler hat sie [für tot erklärt](#), da er keine Zeit mehr für die Weiterentwicklung hat. Glück im Unglück: Ein anderer Entwickler hat sich des Projektes angenommen, und entwickelt die App weiter! Auf der [neuen Projektseite](#) finden sich nicht nur ein paar hilfreiche Informationen; hier kann man auch die .apk Datei herunterladen.

Als Alternative wäre noch links abgebildeter [FTPServer](#) zu nennen. Auch diese App ist "klein und handlich". Zwar wird hier kein eigener Proxy angeboten, dafür gibt es jedoch andere Funktionen: So lassen sich User "chroot"en (d. h. ihr Zugriff wird auf ein ausgewähltes Verzeichnis inkl. Unterverzeichnisse eingeschränkt), der Server auf einzelne definierte WLANs beschränken, u. a. m.. Um einem "Abriss" der WLAN-Verbindung vorzubeugen, kann die App den Standby-Modus deaktivieren.

Wem FTP zu spartanisch ist, der kann z. B. zu [Samba](#) greifen. Der Name geht auf die Bezeichnung des verwendeten Protokolls zurück: SMB ("a" an passenden Stellen einfügen) steht für **S**erver **M**essage **B**lock, vielen auch einfach als "Windows Freigaben" bekannt. Mit diesen "Laufwerken" kann Windows also von Haus aus (und Linux Dank Samba ebenso) prima umgehen, der Androide wird einfach als Laufwerk in das lokale Dateisystem eingebunden. Allerdings mit der rechts abgebildeten App [Samba Filesharing](#) laut ihrer Beschreibung nur das "external Storage", also die SD-Karte. Was mich persönlich irritiert, denn mit den geforderten root-Rechten sollte auch ein Zugriff auf den internen Speicher sowie dessen Freigabe kein Problem darstellen? Wie dem auch sei: Unser eingangs gesetztes Ziel eines "drahtlosen USB-Sticks" ließe sich hiermit umsetzen.



Eine weitere Möglichkeit der direkten Einbindung in das lokale Dateisystem ist mittels [WebDAV](#) gegeben. Von Haus aus unterstützen dies sowohl Linux als auch Mac OS, für Windows muss die entsprechende Client-Software separat besorgt werden. Wie die Einrichtung unter allen drei Systemen vonstatten geht, ist auf der [Website von DavDrive](#) beschrieben. Womit der Name einer Server-App für Android benannt wäre: [DavDrive](#) (links abgebildet) stellt die SD-Karte (und in der Vollversion auch den internen Speicher, diesen allerdings nur im Lesezugriff) per WebDAV zur Verfügung. Die Vollversion lässt sich für knapp einen Euro im Playstore erwerben, und stellt zusätzlich zur Basis-Funktionalität verschlüsselte Verbindungen

(HTTPS), Autostart-Funktionalität, Tasker/Locale Plugin, und einiges mehr zur Verfügung.

Steht die Sicherheit im Vordergrund, muss natürlich auch [SSH](#) Erwähnung finden. Mittels dieses Netzwerkprotokolls lassen sich nicht nur Dateien übertragen; die Möglichkeit eines Terminal-Zuganges (also zur Kommandozeile des Servers) ist ebenfalls gegeben.

Einen passenden Server stellt [QuickSSHd](#) (Abbildung rechts) auf dem Androiden zur Verfügung. Die Implementierung dieser App basiert auf [Dropbear](#), was als besonders ressourcenschonend gilt. Auch wenn [root](#) nicht zwingend vorausgesetzt wird, bringt dieser in diesem Zusammenhang mindestens einen Vorteil mit: Die Verwendung des Standard-Ports 22 wird dadurch möglich, so dass der entsprechende Parameter beim Zugriff (zur Angabe des zu verwendenden Ports) entfallen kann. Leider bietet der Entwickler keine Gratis-Version zum Test an; aber ein Euro ist nun wirklich keine zu große Ausgabe.



Zum Abschluss sei mit [File Expert](#) (linkes Bild) noch ein ganz spezieller Kandidat genannt. Dieser ist weit mehr als nur ein Dateiserver. Man muss sogar soweit gehen, zu sagen: Das macht er nebenbei. Hauptsächlich will die App ein vollwertiger Dateimanager sein – mit allem, was dazu gehört: Kopieren, Verschieben, umbenennen von Dateien und Ordnern, Zugriff auf Daten im Netz per SMB, FTP, SFTP sowie FTPS, Unterstützung für Archive (ZIP/RAR/GZIP/TAR/TGZ/BZ können wie Ordner navigiert und betrachtet werden, ZIP-Dateien lassen sich auch erstellen), Anzeige von Texten und Bildern... Sogar ein App-Manager ist integriert.

Im Umfeld dieser Übersicht sind jedoch die Server-Eigenschaften hervorzuheben. Und hier punktet *File Expert* mit der Bereitstellung sowohl eines FTP-, eines Bluetooth- und eines Webservers. Über alle drei lässt sich im WLAN drahtlos auf die Daten des Androiden zugreifen. Es können Dateien hoch- oder

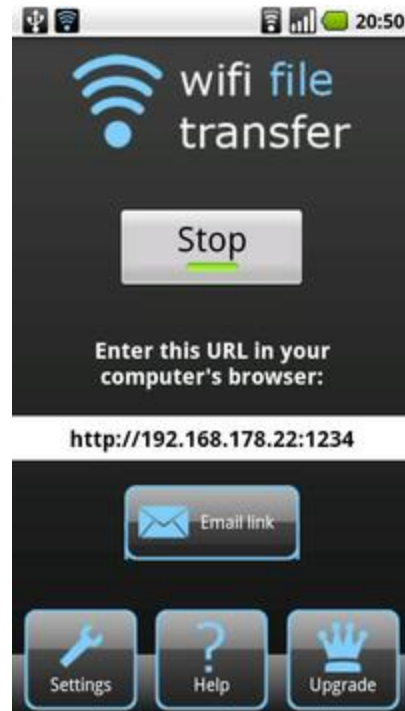
heruntergeladen, Dateien und Ordner umbenannt oder gelöscht werden – alles über ein einfach gehaltenes Interface.

Zu den meisten hier genannten Apps gibt es natürlich noch weitere Alternativen. Sie alle in diesem Kapitel zu beschreiben, würde ein wenig zu weit gehen – daher finden sie sich, wie gewohnt, in einer [Übersicht im Forum](#).

Web-Server

Die Webserver möchte ich grob in drei Kategorien einteilen. Die erste befasst sich, wie bereits die im vorigen Abschnitt beschriebenen Server, mit der Bereitstellung von Dateien – dient also in erster Linie dem Dateiaustausch. In der zweiten Kategorie finden sich sodann Webserver, die vorwiegend Inhalte (statische Webseiten) bereitstellen. Und schließlich soll es auch noch solche geben, die Support für Skriptsprachen wie PHP bieten. Eine Übersicht zu diesen dreien findet sich wieder [im Forum](#).

[WiFi File Transfer](#) (rechtes Bild) stellt das lokale Dateisystem ausschließlich per HTTP im WLAN zur Verfügung. Über eine ansprechende Web-Oberfläche lassen sich Dateien hoch- und herunterladen (auch im Batch-Betrieb, also mehrere auf einmal), umbenennen, kopieren, ZIP-Archive erstellen oder auspacken, und mehr. Für die Betrachtung von Fotos steht sogar eine integrierte Thumbnail-Galerie zur Verfügung. Die Einschränkung der Gratis-Version besteht in einer Limitierung beim Upload: Dateien größer als 4MB werden hier nicht unterstützt (wohl aber in der Kaufversion – die mit 4,9 Sternen eine absolut spitzenmäßige Playstore-Bewertung vorweisen kann).



Auch [Wireless File Transfer](#) (linkes Bild) erlaubt einen Zugriff auf die im Androiden gespeicherten Dateien über das lokale WLAN. In der Kaufversion sind zusätzlich noch ein lokaler Dateimanager, App-Manager und mehr mit an Bord. Die Web-Ansicht ist in beiden Fällen recht ansprechend, und ermöglicht das Anzeigen, Hoch- und Herunterladen, Löschen sowie Umbenennen von Dateien. Darüber hinaus wird auch die Installation von Apps mit einem Klick angesprochen.

Mit [kWS](#) (Bild rechts) begeben wir uns zu den "Content Servern" – also jenen Webservern, die sich um den kompletten Webservice kümmern.

Unter Android beschränkt sich dies zumeist auf sogenannten "statischen Content", also reine HTML-Seiten mit Bildern etc. Die Unterstützung von Skriptsprachen bildet da eher die Ausnahme. [kWS](#) bietet in dieser Hinsicht zumindest [SSI](#) für grundlegende Dinge. Dennoch ist die App mit weniger als 100kB Download ein Leichtgewicht. Und bringt in dem kleinen Paket noch einiges mehr unter: Verzeichnis-Indexe, Resumable File-Downloads,



Zugriffsschutz (Basic und Digest Authentication) – sogar Verzeichnis-Downloads (.tar, .tgz und .zip) und ein integrierter DynDNS Updater sind mit dabei. Bis zu zwanzig parallele Verbindungen sind möglich, auch Protokolldateien (Log-Files) werden erstellt.



Noch mehr bietet die Pro-Version, die sich für weniger als zwei Euro im Playstore erwerben lässt: Diese beherrscht außerdem SSL/TLS für gesicherte Verbindungen (HTTPS), komprimierte Übertragung (gzip), Verzeichnis-Indexe in den Formaten JSON und XML, und bietet weitere zusätzliche Features.

Wer auf SSI verzichten kann, findet in [ServDroid.web](#) (links im Bild) eine Alternative. Diese App nimmt ebenfalls nicht all zu viel Platz weg. In Sachen Funktionsumfang reicht sie an kWS vielleicht nicht heran, muss sich aber auch nicht gerade verstecken: Personalisierte "404 Seiten" ("Dokument nicht gefunden", dito für weitere Fehler), bei eingehendem Request vibrieren, Protokollierung – für die wichtigsten Dinge und ein paar Extras ist auch hier gesorgt. Dankbare Nutzer haben die Möglichkeit einer Donation in Höhe von einem, zwei oder (zusammengenommen) drei Euro.

Zu guter Letzt noch ein Wort zu einem ganz speziellen Kandidaten, der bereits im Kapitel [Androiden vom PC aus verwalten](#) vorgestellt wurde: Der [PAW Server](#) ist ein mächtiges Instrument, der u. a. auch einen Webserver integriert hat. Und selbiger bringt auch Support für Skriptsprachen mit, was zumindest für Webentwickler interessant sein dürfte. Da wäre zum Einen die eigene Skriptsprache, mit der sich auch zahlreiche Systeminformationen darstellen lassen. Zum Anderen wird über ein Plugin zusätzlich PHP unterstützt.

Medien-Server

Zur Abwechslung einmal kein Verweis auf eine Übersicht, da ich an dieser Stelle nur eine einzige App benenne. [Media Show](#) stellt die Galerie des Androiden auf einfache Weise für jedes Gerät im lokalen Netz zur Verfügung, welches über einen Web-Browser mit Adobe-Flash-Unterstützung verfügt. Das sollte auf jeden PC und Mac i. d. R. zutreffen – und so braucht sich nicht der gesamte versammelte Freundeskreis um das kleine Display eines Androiden drängeln, um die Bilder von der letzten Party zu begutachten.



Und wer hat's erfunden? Die Schweizer. Naja, zumindest sitzt das Team dort in der Schweiz, und doktert nach Feierabend an dieser App. Eine gelungene Arbeit, wie ich finde. Die App wird natürlich auf dem Androiden installiert, und stellt die Quelldaten (Fotos und Videos) per WLAN bereit. Die in den Browser einzugebende Adresse wird nach Start des Service in der App angezeigt. Das notwendige Flash wird aus dem Netz nachgeladen, und schon steht dem Vergnügen nichts mehr im Weg!

Doch das ist noch nicht alles: Die Androiden-Galerie lässt sich so auch am Computer bearbeiten. Man kann Fotos herunterladen, online bearbeiten, via Flickr, Facebook und Twitter mit anderen Teilen, und mehr.

TUNING

Einleitung und Überblick

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

Wobei zuerst einmal geklärt werden sollte: 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 da zwei Richtungen: "Schneller, weiter, höher" (Performance/Geschwindigkeit), und "Meiner ist länger als Deiner" (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 der Möglichkeiten:

Generelle Maßnahmen

Toll, dass es Dinge gibt, die beides bedienen: Mehr Speed ohne zusätzliche Akku-Belastung nimmt man gern in Kauf, 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. 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 drauf verzichten kann, sollte das 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 längerer Akku-Laufzeit bei – sondern eher 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 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 persönliche richtige Mischung aus "verfügbarem RAM" und "schnell verfügbaren Apps" festlegen.
- Eventuell Swap-Space nutzen (braucht root)
- Nur für bestimmte Situationen, wo es darauf ankommt: CPU ggf. leicht übertakten. Frisst aber auch mehr Akku. Und benötigt root.

Generelle Tuning-Maßnahmen

Toll, dass es Dinge gibt... ach, das hatten wir schon? Trotzdem schön, dass all die in diesem Kapitel beschriebenen Maßnahmen von jedermann (und -frau, auch -innen bei Bedarf) eingesetzt werden können – von ein paar klitzekleinen Ausnahmen einmal abgesehen. Aber Schluß mit den ewigen Vorreden, und ran an die Bouletten!

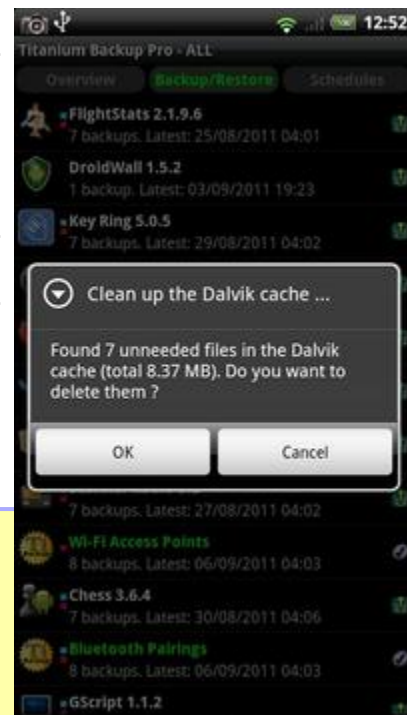
Nicht mehr verwendete/benötigte Apps deinstallieren

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 (ha! Genau, da war auch noch was).

Gerade bei Geräten mit wenig internem Speicher kann das einiges bringen – denn dieser freigewordene 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.

Hat man eine Reihe von Apps auf diese Weise ins Nirwana befördert, kann man sich als Wurzelmann (also Android-User mit root) auch Gedanken um die Bereinigung des Dalvik-Cache machen. Hier sind ja ebenfalls Lücken entstanden, und eine Reorganisation würde für einen schnelleren (und kürzeren, also Akku-schonenderen) Zugriff sorgen. Eine Möglichkeit dazu bietet Titanium Backup Pro mit der Option "Cleanup Dalvik Cache" (rechtes Bild). 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 im Abschnitt [Apps sichern](#) genannte *AppMonster*, welches 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 nein, Werbung lässt sich so nicht entfernen - also Vorsicht, wenn es einen bei selbiger schüttelt...

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.

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

Cache bereinigen

So ein Cache ist schon was 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). Soweit ist das alles prima und überaus nützlich.

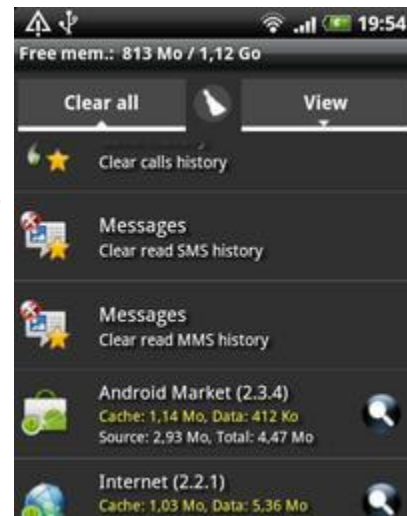
Problematisch wird es erst dann, wenn kein Verfallsdatum festgelegt wurde, und keine automatische Bereinigungsaktion stattfindet - dann sammelt sich im Cache nämlich überwiegend "Hausmüll", den niemand mehr braucht. Der wiederum bremst den Zugriff auf sinnvolle Cache-Daten aus, und belegt wertvollen 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 so 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 hin und wieder 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 Besensschrank? Mit Bordmitteln heißt es da: 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. Keine Übersicht, die vielleicht gar noch nach belegter Datenmenge sortiert wäre.

Bis vor noch gar nicht all zu langer Zeit 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](#) (rechtes Bild) 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 – zukünftig also ganz ohne Klick. 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 und entscheiden, welche App man bereinigen möchte. Nebenbei kann *Quick Cache 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.





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 Einschränkung auf Apps, die überhaupt Cache belegen, beherrscht sie. Und ist darüber hinaus auch noch gratis erhältlich.

Ebenfalls gratis (oder besser: werbefinanziert) gibt es mit [1Tap Cleaner](#) (linkes Bild) eine weitere App, die sich sowohl um den Cache als auch um die anderen "historischen Fälle" kümmert, und dabei auch wieder einen Scheduler mit an Bord hat. Nebenbei kümmert sie sich auch um die sogenannten "App Defaults", und setzt sie bei Bedarf zurück – hilfreich, wenn man beispielsweise einen alternativen Launcher zum Standard gemacht hat und nicht weiß, wie man diesen Standard nun

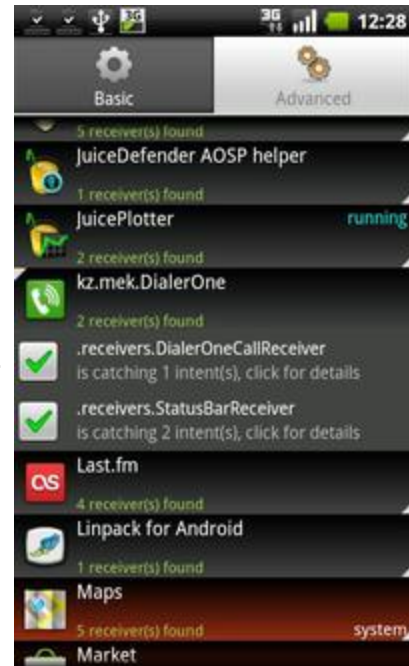
wieder los wird, um den "alten" Launcher nutzen zu können (klar geht das auch in dessen Einstellungen unter *Anwendungen verwalten* – aber warum lange suchen?). Neben diversen Sortiermöglichkeiten lässt sich bei dieser App die Liste der Anwendungen sogar nach Namensbestandteilen filtern. Wer die Gratis-Version ausgiebig getestet hat, und den Entwickler belohnen (oder einfach nur die Werbung loswerden) möchte: Mit einem Euro ist man dabei.

Selten genutzte Apps am automatischen Starten hindern

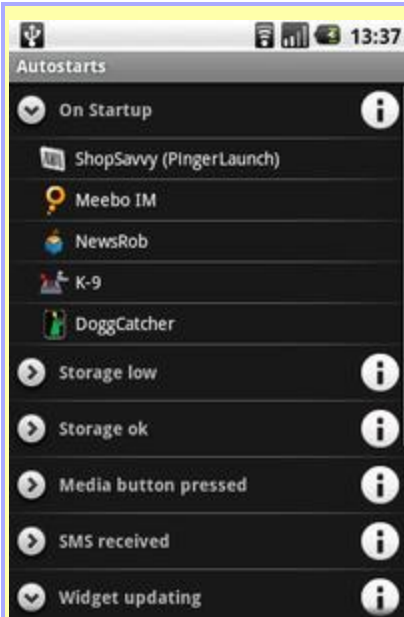
Ein altbekanntes Lied, nicht nur auf unseren geliebten Androiden: So manche App fühlt sich so furchtbar wichtig, dass sie meint, ständig laufen zu müssen. Am besten bereits vor dem Booten. Am schlimmsten sind dabei die ach-so-tollen Apps, mit denen uns Hersteller und Provider "zwangsbeglücken" – und 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. 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 geht leider all zu häufig in die Hose, da 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 (rechtes Bild) 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...



Weitaus bessere Chancen hat man, sofern 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" – wohl vom englischen Wort "intention", Absicht[serklärung], abgeleitet) 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, wie im Screenshot zu sehen, 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 *Autostart 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, zudem es auch noch gratis verfügbar ist. Als Alternative käme eventuell noch **Autostarts** (linkes Bild) in Frage, was 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 wieder herzustellen.

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 selbige eher 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 *LocationService* 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 da ständig läuft und nuckelt – womit sich die Frage meist erübrigt.

Es gibt noch jede Menge weiterer Intents, die sich hinterfragen lassen. Doch sollte dabei bedacht werden: Besser die Finger von Dingen lassen, die man nicht versteht. Schaltet man hier zu viel ab, funktioniert eine App u. U. nicht mehr richtig – und das ist sicher nicht Ziel der Übung.

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

Um nun 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.

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...



Das Märchen vom Task-Killer

Viele "alte Hasen" schwören darauf: Task-Killer seien das ultimative Mittel, um das System flüssig und den Strombedarf niedrig zu halten. Von genügend anderen Anwendern, die definitiv tiefgreifendes Wissen haben, hört man genau das Gegenteil – teilweise sogar bis zu dem Extrem, dass man "Task-Killer gar nicht mehr benötigt, da kümmert sich Android doch selbst drum".

Da haben wir sie also nun: Zwei Statements, die verschiedener nicht sein könnten – beide von Leuten mit viel Erfahrung. Wem soll nun Glauben geschenkt werden? Da liegt schon fast auf der Hand, dass die Antwort irgendwo in der Mitte liegen muss, also beide Aussagen einen gewissen Wahrheitsgehalt haben.

Es ist durchaus richtig, dass Android sich selbst um den Speicher kümmert – und bei Bedarf nicht mehr benötigte Apps aus selbigem herauswirft. Dafür gibt es den [OOM-Killer](#): Wird der Speicher knapp (ist das System also Out-Of-Memory), tritt er in Aktion. Aber nur dann. Läuft hingegen eine App Amok, und verwandelt die CPU in eine Warmhalteplatte für den Kaffeebecher, fühlt sich der OOM-Killer davon überhaupt nicht angesprochen – solange besagte Amok-App nicht zeitgleich sämtlichen Speicher auffrisst. Dies wäre also ein Fall für den "klassischen Task-Killer".

Die Frage ist also weniger, *ob*, als vielmehr *wann* man einen Task-Killer einsetzen sollte:

- Falsch: "ich will den Speicher freiräumen". Dafür ist der "OOM Killer" zuständig (siehe [RAM bereinigen](#))
- Falsch: "ich will Akku sparen". Das geht mit einem Task-Killer i. d. R. eher nach hinten los.

- Richtig: "eine App hat sich aufgehängt, und blockiert [irgendwas]". Hier ist der Task-Killer angesagt – weil bis der "OOM-Killer" hier zuschlagen würde... Und ein Reboot ist nicht gerade die wünschenswerte Alternative.
- Richtig: "eine App läuft Amok" (Panik-Mode: Man erwischt gerade eine App dabei, wie sie alle persönlichen Daten inkl. Nackt-Fotos auf eine berühmte Website hochlädt...). Oh ja: Abschießen! Oder gleich abschalten. Weil: Bis der OOM-Killer... genau, da ist es dann eh zu spät...
- Notfalls richtig: Kurz nach Betätigung des Start-Knopfes einer App (wann sonst?) fällt einem auf: Das würde jetzt zu lange laufen, weil [xxx]. Richtigerweise hätte man dann besser gar nicht erst den Knopf gedrückt. Lässt sich die App nicht anders vom "Weitermachen, bis der Arzt kommt" abbringen, und belegt dabei wertvolle Ressourcen (CPU, Netztraffic), ist das "Abschießen" mittels Task-Manager auch hier die einzige Alternative zum Reboot.

Die generelle Aussage, dass Task-Manager ein tolles Mittel zum Akku-Sparen seien, fängt jedoch definitiv mit "Es war einmal..." an. Für Android-Versionen, die vor dem Punkt nur eine "1" stehen haben, traf das vielleicht tatsächlich zu. Aktuell ist es jedoch wesentlich häufiger der Fall, dass eine frisch gekillte App binnen weniger Sekunden schon wieder aktiv ist – was bei "automatischen Task-Killern" zu einem ewigen Kreislauf aus "Start-Kill-Start" führen würde, und somit alles andere als ressourcensparend ist.

Längere Akkulaufzeiten erreichen

Kürzlich absolvierte ein Bekannter seinen zweiwöchigen Auslandsurlaub. Da er dort eine lokale SIM-Karte einsetzen wollte, um Roamingkosten zu vermeiden, benötigte er ein Reserve-Handy – um auch unter seiner heimischen Rufnummer erreichbar zu sein. Zu diesem Zweck wurde ein "Billig-Telefon" der Marke Nokia hergenommen und aufgeladen. Das Ladegerät hätte er ruhig zu Hause liegen lassen können: Obwohl nur im Flugzeug abgeschaltet, und ansonsten permanent in Bereitschaft, hielt es mit der einen Akkuladung die vollen zwei Wochen durch, einschließlich einiger kürzerer Telefonate.

Zwei Wochen mit einer Akku-Ladung! So manch einer wäre schon froh, wenn er es auf zwei Tage bringen würde! Sind unsere Akkus schlechter geworden, oder woran liegt es?

Wohl eher daran, dass moderne Smartphones mittlerweile vollwertige Computer im Hosentaschen-Format sind. Was musste denn das oben genannte Nokia "leisten"? Kurz gesagt: "GSM Standby", und ein paar GSM Gesprächsminuten. Mehr nicht. Unsere Smartphones sind hingegen permanent online, surfen am liebsten mit 3G, und spielen parallel dazu Youtube-Videos ab. Wer nun einen kurzen Blick in die passende [Übersicht im Anhang](#) wirft, sieht den Unterschied deutlich: Würden wir unsere Androiden auch auf selbigen Standby reduzieren, und sämtliche "überflüssigen" Dienste deaktivieren, sollten wir eigentlich ebenfalls die Wochenfrist überschreiten können.

In "voller Ausbaustufe" ist das sicherlich wenig sinnvoll – man hat sich so ein Smartphone ja unter anderem wegen all dieser "Extras" gekauft. Aber gibt es vielleicht Kompromisse? Schließlich fährt ja auch keiner seinen Porsche im fünften Gang mit 220 km/h in die Garage, nur weil er ihn für "schnelles Fahren" gekauft hat. Ebenso wenig telefoniert man non-stop, und schaut auch nicht 24 Stunden pro Tag Youtube-Videos über die 3G Verbindung. Da fehlt dem Gerät wohl ein wenig Intelligenz: Es muss doch *fühlen*, wann wir etwas brauchen und wann nicht... Aber dann hieße es wohl "*die* Smartphone", und nicht "*das*". Also müssen wir dem kleinen Androiden ein wenig männliches Macho-Gehabe (meiner ist größer, schneller, weiter) ab- und weibliche Intuition angewöhnen. Möglichkeiten dazu sollte es doch geben. Und nein: Lippenstift, Lidschatten und Maskara helfen dabei nicht...

Ob da gerade etwas überdimensional Strom zieht, während der Androide beispielsweise eigentlich "ungenutzt" in der Tasche steckt, verrät die App [DrainGuard](#) (Abbildung rechts). Sie verrät dem Anwender zwar nicht, wer der Schuldige ist – doch zumindest warnt sie bereits, wenn der Akku-Sauger am Werk ist, und nicht erst, wenn der Akku bereits leer gesaugt worden ist. Die notwendigen Werkzeuge zum Aufspüren des Übeltäters finden sich im Kapitel [System-Info](#).



Deaktivierung ungenutzter Dienste

Unschwer in der [im Anhang befindlichen Übersicht](#) erkennbar: Die Daten-Dienste gehören (nach der Video-Wiedergabe) zu den größten Verbrauchern. An dieser Stelle ließe sich also am besten sparen – sofern man auf etwas verzichten kann. Was das vielleicht sein könnte, ist wiederum von Anwender zu Anwender unterschiedlich: Als erstes gilt es daher, das eigene Nutzungsverhalten zu analysieren.

Wer über kein einziges Bluetooth-Gerät verfügt, kann Bluetooth komplett deaktivieren. Im Standby macht das zwar (nach GPS Standby) am wenigsten aus, aber auch Kleinvieh macht Mist. Das passende Häkchen findet sich in den System-Einstellungen unter *Drahtlos & Netzwerke*.

Auf der gleichen Seite findet sich mit *Mobilfunknetze* ein Untermenü, das ein Häkchen für *Nur 2G Netzwerke* anbietet – und somit 3G komplett abschaltet. Das bietet ein weit größeres Einsparpotential, sofern man auf die schnellere Datenübertragung zum größten Teil verzichten kann; für die "kleinen Dinge" wie Mail und die Synchronisation der meisten Apps (insbesondere Kalender und Adressbuch) ist 2G definitiv ausreichend. Weniger interessant ist diese Möglichkeit hingegen für diejenigen, die viel mit dem Webbrowser oder der Youtube-App unterwegs sind, ohne dabei auf WLAN-Netze zurückgreifen zu können – da ein Umschalten über dieses Unter-Unter-Menü auf Dauer doch ein wenig umständlich ist. Und spätestens seit Android 2.3 (Gingerbread) ist ein direktes Umschalten per Widget ohne root leider auch nicht mehr möglich.

Ähnliches gilt für GPS, welches aber glücklicherweise im Standby so gut wie gar keinen Strom verbraucht. Sobald allerdings eine App zur Standortbestimmung darauf zugreift (erkennbar am GPS-Symbol in der Statusleiste – siehe Beispiel-Symbole rechts), macht sich auch das deutlich bemerkbar. Wird GPS vom Anwender selbst nur selten genutzt (sagen wir, maximal an einem Tag im Monat, oder nur im Urlaub), schaltet man dieses ebenfalls besser komplett aus – spätestens, wenn man eine dieser werbefinanzierten Apps beim Zugriff erwischt. Das passende Häkchen findet sich im Menü *Standort & Sicherheit*.



Bleibt noch unser bester Freund: Das WLAN. Wer das gar nicht braucht... der gehört mit Sicherheit einer Minderheit an. Glücklicherweise kann man diesen Dienst nach Belieben mit diversen [Schnellumschaltern](#) schnellumschalten; bei einigen Geräten (und [Custom-ROMs](#)) ist der passende "Knopf" überdies bereits in der Notification-Area integriert.

Ist zwar nicht direkt ein Dienst, passt aber mit in diese Reihe: Auch die Kamera ist ein besonderer Schluckspecht. Die Kamera-App sollte man daher besser nicht nach Benutzung im Hintergrund weiterlaufen lassen, sondern beenden. Das gilt auch für diverse Barcode-Scanner, die ja auch die Kamera benutzen: Sollte sich ein solcher nicht richtig beenden wollen (was man meist am kontinuierlich starken Akku-Verbrauch erkennen kann), ist wohl ausnahmsweise einmal der Einsatz eines [Task-Killers](#) gerechtfertigt.

Display-Einstellungen

Diese bieten sich ebenfalls für deutliche potentielle Einsparungen an: Ein voll aufgedrehtes Display verbraucht wesentlich mehr Strom als ein 2G-Telefonat, während es auf niedrigster Stufe nur gut halb soviel benötigt wie besagtes Gespräch. Hier empfiehlt es sich daher, einen guten Kompromiss zu finden – zum Einen die Helligkeit, und zum Anderen das Timeout (nach welchem Zeitraum der Benutzer-Inaktivität das Display abgeschaltet werden soll) betreffend.

Beides lässt sich mit Bordmitteln im Menü *Display* konfigurieren. Bequemer geht auch dieses über diverse [Schnellumschalter](#) – besonders, wenn bei direkter Sonneneinstrahlung ohnehin kaum noch etwas auf dem Bildschirm zu erkennen ist. Die *Automatische Helligkeit* ist leider meist ein wenig zu hell (und damit stromhungrig); die "beste Einstellung" ist jedoch sehr subjektiv, und muss daher individuell ermittelt werden.

Hintergrunddaten und Synchronisierung

Dieses Thema wurde ja bereits im Kapitel [Datensynchronisation im Hintergrund](#) angesprochen: So manche App möchte gern ständig Daten austauschen. Nicht immer ist dies aber nötig: Die wenigsten aktualisieren etwa ihre Kontaktliste mehrmals stündlich, sei es am Androiden oder am heimischen Bürorechner. Was nicht oder nicht ständig aktualisiert werden muss, kann man daher einfach im Menü [Konten & Synchronisierung](#) abschalten. Ein manueller Abgleich aus der betreffenden App heraus ist in der Regel immer noch möglich.

Nicht jede App trägt sich jedoch an dieser Stelle ein. Bei den meisten lässt sich aber zumindest das Intervall konfigurieren (andernfalls ist es vielleicht ratsam, sich nach einer Alternative umzusehen, bei der dies der Fall ist). Für RSS-Reader reicht in den überwiegenden Fällen ein Intervall von einer Stunde oder höher (zum Einsparen des Datenvolumens kann man das darüber hinaus auch noch auf bestehende WLAN-Verbindungen einschränken).

In Sachen Mail empfiehlt sich, sofern möglich, die Verwendung von IMAP Push gegenüber einem ständigen "Pull": Dabei fragt nicht der Client ständig nach "Gibt es neue Mail?" (das wäre "Pull" oder neudeutsch "Pollen"), sondern der Server sagt im Falle des Falles von sich aus Bescheid. Die Information kommt daher im Regelfall sogar früher. Auf den Energieverbrauch bezogen gilt diese Empfehlung allerdings nur bedingt, da auch bei IMAP Push die Verbindung ab und an einen "Refresh" benötigt. Grob gesagt: Wer bei Eingang einer Mail sofort reagieren muss, ist damit gut dran. Genügt jedoch eine stündliche Aktualisierung, ist das altgewohnte "Pollen" sparsamer.

Viele Apps berücksichtigen auch eine System-Variable, die kundtut, ob eine Datensynchronisation im Hintergrund zur Zeit erwünscht ist (die Android-API ist hier leider schwammig: Statt die Berücksichtigung verbindlich zu machen ("must"), heißt es dort nur "[sollte berücksichtigt werden](#)" ("should"). Markanterweise gehören zu den Missachtern an erster Stelle Apps wie [GMail](#) und [Latitude](#) – Google geht also wieder einmal mit "gutem Beispiel" voran. Andere wie beispielsweise [WhatsApp folgen diesem](#).

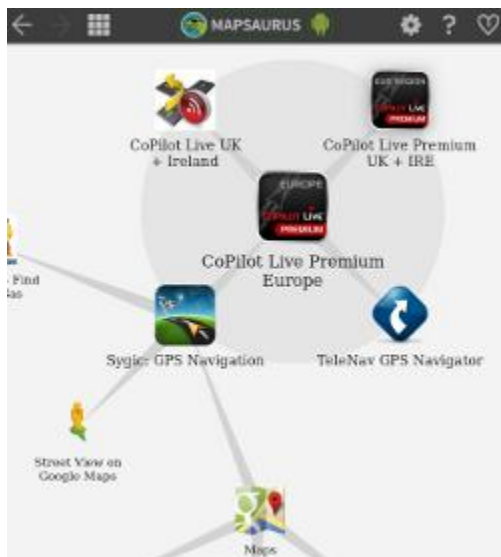


Für Apps, die diese System-Variable berücksichtigen, greifen dann wieder die kleinen [Schnellumschalter](#), wie das rechts abgebildete [Dazzle](#): In der Abbildung wäre es der zweite Button von rechts, über den mit einmal Antippen die Synchronisation unterbunden (und natürlich zu einem beliebigen späteren Zeitpunkt wieder erlaubt) werden kann.

Weitere interessante Überlegungen zum Thema Hintergrunddaten finden sich bei [SE-World](#) – obwohl der Artikel bereits ein wenig älter ist, bleiben die Informationen interessant (und sind m. E. noch nicht veraltet oder überholt).

Verschiedenes

Dann wären da noch verschiedene Dinge, die so selbstverständlich sind, dass man gar nicht daran denkt. Etwa, dass man energiehungrige Apps nicht unbedingt im "Hintergrund" weiterlaufen lässt – sondern sie beendet. Klar, eigentlich sollten sie im Hintergrund ja keine Ressourcen belegen; aber eigentlich gibt es das Wort "eigentlich" auch nicht. Denken wir beispielsweise an eine Navigations-App: Diese aktualisiert i. d. R. auch im Hintergrund permanent den Standort – bei Verwendung von GPS nicht gerade eine Akku-schonende Tätigkeit. Auch so mancher Barcode-Scanner hält im Hintergrund die Kamera auf Trab. Wieder andere Apps pflegen im Hintergrund fleißigen Netzwerkverkehr, und so manche "Gratis-App" lädt auch alle paar Sekunden ein neues Werbebanner, selbst wenn es niemand zu Gesicht bekommt. Ergo: Apps, die man nicht mit voller Absicht im Hintergrund behalten möchte (etwa, weil man sie gleich wieder zu verwenden gedenkt) nicht über die "Home-Taste" (die mit dem Häuschen), sondern – sofern verfügbar – direkt über die "Beenden"-Funktion (häufig über die Menü-Taste erreichbar), oder zumindest über die "Zurück"-Taste verlassen. Bei hartnäckigen Verweigerern, die sich als Stromfresser herausstellen, ist dann ausnahmsweise einmal der Einsatz eines [Task-Killers](#) gerechtfertigt.



Bei Apps, die permanent im Hintergrund laufen wollen (und sich partout nicht beenden lassen), obwohl der Anwender das nicht benötigt, sollte man sich nach Alternativen umsehen. Nicht ersetzbare Apps sind selten, häufig gibt es mehrere Apps, die das gleiche können. Guter Startpunkt für eine Suche ist die zur App gehörige Seite bei AndroidPIT: Dort sind in der Regel handverlesene Alternativen direkt unterhalb der Appbeschreibung genannt. Auch die [Große Sammlung der App-Übersichten nach Einsatzzweck](#) kann sich als hilfreich erweisen. Ebenfalls einen Blick Wert ist in diesem Zusammenhang die App [Mapsaurus](#), oder auch die [Website](#) (linkes Bild): App angeben – und schon tauchen passende Alternativen

(mit zugehörigem Playstore-Link) auf. Wählt man diese nun an, geht das Spielchen weiter. Da sollte sich doch etwas finden lassen! Sehr begrüßenswert wäre es allerdings, wenn sich der zuständige Entwickler auf Anfrage des Problems annimmt – und eine Lösung liefert...

Nicht zu vergessen: Widgets. Hinter nahezu jedem Widget verbirgt sich ein zugehöriger Hintergrund-Prozess, der dieses mit Daten versorgt. Woher soll es sonst über das aktuelle Wetter, die Anzahl laufender Apps, die Speicher-Belegung, und all diese Dinge Bescheid wissen? Der Einsatz von sich automatisch aktualisierenden Widgets sollte daher sparsam und mit Bedacht erfolgen. Was man davon nicht benötigt: Runter vom Desktop. Die eventuell zugehörige App steht weiterhin unabhängig aus dem App-Drawer heraus zur Verfügung. Und bei Bedarf lässt sich das Widget jederzeit wieder hinzufügen.



Einen hab ich noch: So toll, cool, und was-auch-immer sie erscheinen mögen: Live-Wallpaper sind besondere Energiefresser. Sie verbrauchen permanent CPU, meist auch noch weitere Ressourcen, und somit anständig Strom. Also besser darauf verzichten, solange einem die Akku-Laufzeit nicht egal ist.

Wer jetzt das "große Fragezeichen" im Gesicht hat, wie denn "Bösewichter" wohl ausfindig zu machen seien: Ein Blick in das Kapitel [Monitoring](#) zeigt da einige Möglichkeiten auf. Auch ein Blick in das Kapitel [Ausführlichere System-Infos](#) fördert hilfreiche zu Tage.

Akku-Pflege

Nix mit "Waschen, Schneiden, Legen", nein – Akkus mögen kein Wasser. Aber sie mögen es, wenn man sie gut behandelt. So kann die Beherrschung einiger kleiner Tipps nicht nur zu einer besseren Laufzeit im aktuellen Ladezyklus, sondern auch zu einer höheren Lebensdauer des kleinen Kraftwerks beitragen. Nichts kompliziertes, auch spezielles Zubehör braucht es nicht. Vielmehr geht es um das "Gewusst-Was" und "Gewusst-Wie". Da ist für jeden etwas dabei.

Zusätzliche Informationen (auch zum Umgang) lassen sich ebenfalls der Wikipedia entnehmen: Sowohl für [Lilo-Akkus](#) als auch für [LiPo-Akkus](#). Eine kurzgefasste Aufstellung der wichtigsten Fakten findet sich auch bei [Sinnvoll-Online.DE](#).

Temperatur

Aktuelle Androiden verwenden sogenannte Lithium-Ionen-Akkus (Lilo; einige setzen auch auf die Weiterentwicklung Lithium-Polymer (LiPo), wie Motorolas *Droid Razr* oder auch *CLIQ*; ebenso eine ganze Reihe Tablets: *Acer Iconia*, *Asus Transformer*, *Samsung Galaxy Tab*, u. a. m. – für diese Geräte gilt das folgende im Groben aber ebenso; platt gesagt, sind LiPo-Akkus kompakter, dafür aber auch empfindlicher was den Temperatur-Bereich betrifft). Diese mögen es nicht gern so warm: Also nicht in die Sonne legen (sie werden ohnehin nicht braun), auch die Hosentasche ist nicht der ideale Aufbewahrungsort. Bevor jemand fragt: Nein, der Kühlschrank auch nicht. Lieblings-Temperatur von Lilo-Akkus: Um die 20°C. Unterhalb von 10°C lässt die Leistungsfähigkeit nach, der "Arbeitsbereich" wird meist mit 0..40°C angegeben.

Laden / Entladen

Eine vollständige Entladung bis zur Selbstabschaltung des Gerätes sollte zwar hin und wieder zur [Kalibrierung](#) stattfinden – aber nicht zu häufig: Sowohl Lilo als

auch LiPo Akkus mögen eine vollständige Entladung nicht so gern. Idealerweise schließt man sie also wieder an das Ladekabel an, bevor der Ladestand die 50% Marke unterschreitet. Höhlen-Taucher wissen, wovon ich Rede, wenn ich hier jetzt den Begriff "Reserve" in den Raum stelle: Natürlich kann man auch die "unteren 50%" nutzen – gesünder (in diesem Fall für den Akku) ist es jedoch, darauf nicht angewiesen zu sein. Einen Memory-Effekt kennen weder Lilo noch LiPo, diesbezüglich ist daher nichts zu befürchten.

Den Akku kalibrieren

Der Ausdruck ist ziemlich irreführend. Genau genommen wird nicht etwa der Akku, sondern der Androide kalibriert. Letzterer lernt dabei nämlich, wie es um die tatsächliche Akku-Kapazität bestellt ist, und kann seine Statistiken entsprechend anpassen. Damit wird vermieden, dass das Gerät "plötzlich und unerwartet" bei einem angeblichen Akku-Stand von über 30% "das zeitliche segnet". Oder aber die Anzeige schnell von 100% auf 1% fällt, um dann noch ein paar Stunden dort stehen zu bleiben – bevor der Akku wirklich leer ist.

Dies geschieht durch einen sogenannten "vollständigen Ladezyklus". Bildlich gesprochen, wird der Akku dabei einmal von 100% auf 0% und wieder auf 100% gebracht – wobei während des gesamten Zyklus das Ladegerät exakt ein Mal mit dem Gerät verbunden wird (oder auch umgekehrt; auf jeden Fall eine volle Ladung ohne Unterbrechung). Optimalerweise werden zu Beginn dieses Zyklus noch die Akku-Statistiken gelöscht, was in der Regel jedoch root-Rechte voraussetzt. Doch dieser Schritt scheint eher einem Mythos gezollt, wie Android-Entwicklerin Diane Hackborn laut **XDA-Developers** erklärt: In der batterystats.bin Datei werden laut ihrer Aussage keine Daten der Batterie, sondern ausschließlich die (auch unter *Einstellungen* → *Akku-Verbrauch* einsehbaren) Verbrauchsdaten gespeichert. Somit erübrigt sich eigentlich auch der Einsatz der folgenden Apps, mit deren Vorstellung ich dem Mythos dennoch meine Referenz erweisen möchte:





Hilfreich bei der Kalibrierung sind Tools wie das rechts oben abgebildete [Battery Calibration](#). Diese App führt den Anwender Schritt für Schritt durch den Prozess, und übernimmt nebenbei auch die Löschung der gespeicherten Akku-Statistiken aus dem Android-System.

Das Vorgehen ist dabei total einfach: Man startet die App, und steckt das Ladekabel an. Anschließend stellt man sicher, dass beide Buttons jeweils ein "grünes Lämpchen" enthalten – womit die App angewiesen wird, bei Erreichen einer vollständigen Aufladung (100%) den Anwender mit Piepsen zu benachrichtigen, dass es weitergehen kann. Sobald dieser Zeitpunkt erreicht ist, betätigt man den großen Button in der Mitte (das löscht die `batterystats.bin` aus dem System), und startet den Androiden neu (womit die genannte Datei neu angelegt wird). Ladekabel raus, und einmal

komplett leerlaufen lassen (mindestens bis 20%, besser weniger); anschließend Ladekabel wieder anschließen, und in einem Rutsch wieder vollständig aufladen. Fertig.

Wichtig ist, wie beschrieben, ein vollständiger Zyklus – da könnte man das also auch umgekehrt machen. So tut das etwa [Battery Calibrator](#) (linkes Bild), und startet am "Nullpunkt des Seins". 20% Ladung oder weniger, gerade genug, um die `batterystats.bin` Datei noch vor dem Durchstarten löschen zu können, werden als Anfang des Prozesses gewertet. Das Vorgehen ist ansonsten identisch: Die Datei wird gelöscht, das System neu gestartet, und in einem Zug einmal auf 100% und zurück gebracht.

Wo man beginnt, bleibt dem Anwender überlassen – das Ergebnis sollte in beiden Fällen identisch sein: Eine korrekte Ladestands-Anzeige. Nur für das Löschen der Statistik-Datei benötigt es i. d. R. root, die genannten Apps vereinfachen lediglich den Prozess.

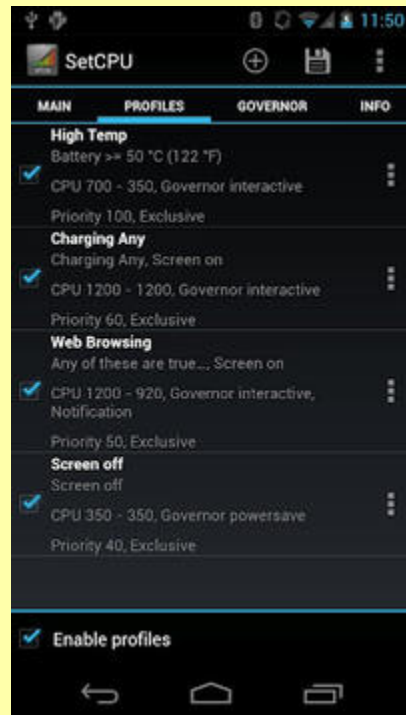
CPU untertakten

Untertakten? Wollen nicht alle in die andere Richtung? Klar, das kann im Winter ganz praktisch sein: Zusätzlich 3G an, heftig Downloaden, HD-Videos schauen, und das Ladekabel dabei angeschlossen – diese Kombination ergibt einen wunderbaren Fingerwärmer. Der aber schnell erkaltet, sobald das Ladekabel abgezogen wird – weil der Akku das dann nicht lange mitmacht...

Und jetzt bin ich dran mit der Frage: Wollten wir nicht in die andere Richtung, also den Akku zu möglichst langer Laufzeit animieren? Ein geringerer Prozessor-Takt kann dazu beitragen. Nicht umsonst wird in den Medien bereits über eine 5-Kern-CPU für Smartphones theoretisiert: Vier Kerne mit ordentlich Power, und der fünfte mit geringerer Taktrate sowie weniger Power für die Hintergrundaktivitäten bei abgeschaltetem Bildschirm. Das bekommen wir softwareseitig sicher nicht so ganz hin – können uns dem aber zumindest annähern. Root vorausgesetzt.

Beispielsweise unter Zuhilfenahme des rechts abgebildeten [SetCPU](#) und entsprechenden Profilen: Wann immer die volle Leistung nicht benötigt wird, kann die CPU gedrosselt werden. Etwa generell bei abgeschaltetem Display. Oder während unserer Nachtruhe. Empfehlenswert ebenfalls, sofern ein bestimmter Ladestand des Akkus unterschritten wird (damit man es noch zur nächsten Ladestation schafft, ohne dass das Gerät abschaltet).

Eine Steuerung der CPU ist allerdings auch in diverse andere "Automaten-Apps" integriert. Natürlich in [Tasker](#) – aber auch in einige der [kleinen Helferlein](#), die im Folgenden kurz vorgestellt werden.



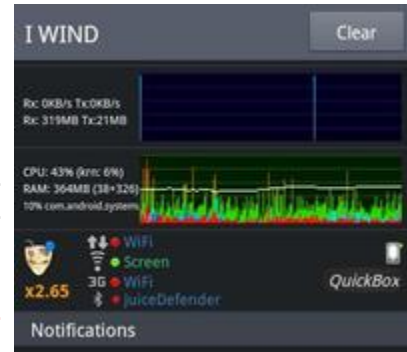
Kleine Helferlein

[Menehune](#)? [Heinzelmannchen](#)? Eine gewisse Ähnlichkeit lässt sich nicht leugnen: Die Apps, um die es hier geht, bekommt man während ihrer Arbeit nicht zu Gesicht. Sie arbeiten flink, und können wahre Meisterwerke vollbringen (wie beispielsweise, ein Smartphone mit einer einzigen Akkuladung über mehrere Tage am Laufen zu halten). Der Unterschied: Unsere Helferlein sind auch heute noch am Werkeln. Und ihre Erfolge lassen sich überprüfen und belegen. Außerdem haben sie im Forum von AndroidPIT eine [eigene Übersicht](#).

Apps für den "Handbetrieb" wurden ja bereits bei den [Schnellumschaltern](#) vorgestellt. Zeitschaltuhren, orts- und eventbasierte Schalter etc. gab es im Kapitel [Automatisierung](#). Bleibt denn da überhaupt noch etwas übrig? Aber klar

doch: Die App-Automaten, die sich speziell dem energiesparenden Thema gewidmet haben: Save Battery!

Der "SaftVerteidiger" ist die erste App, die man in Foren zu diesem Thema genannt bekommt. Und sie ist sicher auch die umfangreichste und flexibelste App – zumindest in der Ultimate-Version. JuiceDefender (rechtes Bild: In der Notification-Area) kommt nämlich in drei Ausbaustufen daher: Einer kostenlosen, einer gehobenen (Plus), und einer Deluxe (Ultimate) Variante. Und natürlich unterscheiden sich die drei Versionen in ihrem Funktionsumfang, wie [diese Übersicht auf der Homepage](#) zeigt.



Alle drei Versionen bieten Widgets (siehe linkes Bild) sowie Notifications (rechts oben), und bieten darüber hinaus mindestens zwei Profile an (normal / aggressiv; extreme / customized kommen ab der Plus-Version dazu). Alle Versionen erlauben die Kontrolle der mobilen Daten (ab der Plus auch WLAN, und mit Ultimate zusätzlich Network Switching, AutoSync und Bluetooth). Die Hintergrund-Daten-Synchronisierung lässt sich mit allen drei Versionen steuern (spezielle nächtliche Schedules gibt es dann ab der Plus, und mit Ultimate zusätzlich Schedules für Peak-Hours und Wochenende). Dinge wie Display-Timeout und -Helligkeit sowie GPS-Kontrolle gibt es nur in der Ultimate, mit Ultimate und root lässt sich sogar die CPU steuern. Eine Sonderbehandlung einzelner Apps ist ab der Plus Version verfügbar.

Das waren jetzt ein Stapel Details. Abschließend noch einige Worte zur allgemeinen Funktionalität: Je nach Bedarf regelt *JuiceDefender* die entsprechenden Verbraucher. So lässt sich beispielsweise die Netzwerkverbindung "Stottern" (d. h. in Intervallen aktivieren und deaktivieren; fürs mobile Netz geschieht dies durch Umbenennen des APN), wobei (ab der Plus-Version) ausgewählte Anwendungen auch permanente (oder gar keine) Aktivierung für ihre Übertragungen gewährt bekommen können. Wird ein Minimum an Ladezustand unterschritten, kann die App den Netzwerkverkehr auch komplett unterbinden. Ab der Plus-Version ist es überdies möglich, WLAN auch ortsbezogen zu aktivieren.

Bereits die Gratis-Version verspricht eine signifikante Laufzeitverlängerung, mit der Plus-Version sind für ca. zwei Euro die meisten Ansprüche abgedeckt. Für diejenigen, die ein Maximum herausholen möchten, gibt es schließlich die Ultimate-Version für ca. fünf Euro. In allen Fällen wird jedoch die Gratis-Version als "Unterbau" benötigt: Plus und Ultimate sind lediglich AddOns.

Als Alternative zu *JuiceDefender* wird häufig **Green Power** (rechtes Bild) genannt – beide Apps sind auch durchaus vergleichbar. Die meisten Dinge werden von beiden Apps gleichermaßen unterstützt, wenn sich ihre Verfügbarkeit auch unterschiedlich auf die Gratis- und Bezahlversion(en) verteilt. So ermöglicht *Green Power* bereits in der kostenlosen Version, mobile und WLAN Daten basierend auf Schedule, Display Status, Ladegerät, Signal Level, und weiteren Kriterien zu steuern (Signal-Level heißt hier: Wird die Verbindung zu schwach, etwa in der U-Bahn, wird sie abgeschaltet – und in Intervallen geprüft, ob sich eine Aktivierung wieder lohnt). Dafür sind Widget und Nachtmodus erst in der Bezahlversion (für anderthalb Euro) verfügbar, ebenso wie Bluetooth-Schaltung. Hinzu kommt in der Premium-Version auch Unterstützung für **Tasker** – womit sich die fehlende ortsabhängige WLAN-Schaltung und CPU-Steuerung dann umsetzen ließe. Notfalls auch ohne *Green Power*.



Eine automatische Steuerung verspricht auch **SuperPower** (linkes Bild, in der Notification-Area). Hier werden mobile Daten, WLAN, Bluetooth, CPU (root), 2G/3G (root), GPS (root), AutoSync und Hintergrunddaten (root) basierend auf Kriterien wie dem Status von Display/WLAN/LockScreen/Tethering/Schlafmodus/

Ladezustand/Batteriestand oder auch Download Speed und Vordergrund-App gesteuert. Auch diverse Schedules lassen sich nutzen (etwa Nachtmodus, Daten-Stottern). Dinge wie GPS-Kontrolle oder auf Signalstärke basierende Schaltung sind in Planung.

Battery Saver (rechtes Bild) verbindet das automatische Management mit Profilen. So kümmert sich die App beispielsweise im Hintergrund darum, bei zu schwachem Signal WLAN bzw. Radio zu deaktivieren (und periodisch auf Verbesserungen zu prüfen, um es ggf. wieder zu aktivieren). Darüber hinaus lassen sich verschiedene Profile (etwa für die Arbeit, zu Hause, draußen...) mit speziellen Einstellungen für Bildschirmhelligkeit, Klingelton-Lautstärke, u. a. m. definieren. Auch das (Ent-)Ladeverhalten lässt sich anhand einer entsprechenden Grafik nachvollziehen.

Es ließen sich noch eine ganze Reihe weiterer Apps aufzählen, die einzelne Aspekte wie die automatische WLAN-Umschaltung u. a. m. abdecken. In der eingangs genannten Übersicht sind sie aufgeführt – hier würden sie einfach den Umfang des Buches sprengen.



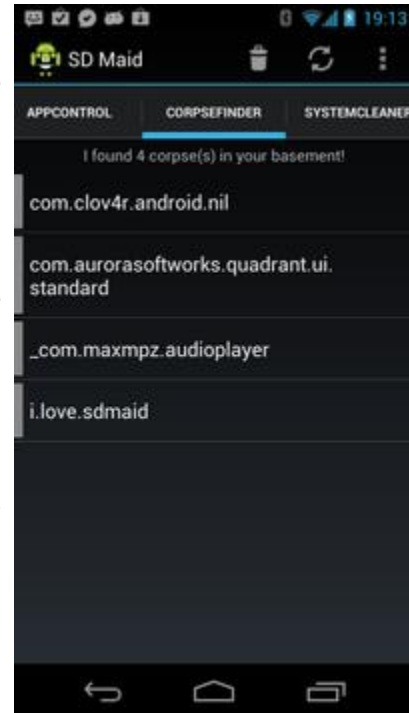
Platz im internen Speicher schaffen

Aufräumen

Irgendwie naheliegend: Wenn man etwas löscht, wird Platz frei. Besonders geeignete Kandidaten dafür sind Apps, die man ohnehin nicht mehr benötigt – was ja bereits in [diesem Kapitel](#) beschrieben wurde (genau wie für root-User das Säubern des Dalvik-Caches). Die [Bereinigung von Cache, Chroniken & Co](#) war ebenfalls bereits genannt.

Doch so manche App hinterlässt auch nach ihrer Deinstallation noch die eine oder andere Leiche im Keller. Um diese aufzuspüren und zu beseitigen, bietet sich ein Blick auf [SD Maid](#) (rechtes Bild) an. Damit lassen sich gezielt Rückstände aufspüren und entfernen.

So ganz nebenbei kümmert sich *SD Maid* mittlerweile auch um die anderen gerade genannten Dinge: Installierte Apps lassen sich (zusammen mit dem von ihnen belegten Speicherplatz) auflisten, um unnötige Exemplare zu entsorgen. "Duplicates" zeigt doppelte Dateien auf. Mit dem "Searcher" lassen sich Dateien finden, deren Name auf ein bestimmtes Muster passt. Um Cache, Log-Dateien, und dergleichen mehr kümmert sich wiederum der "System Cleaner". *SD Maid* scheint also für den Frühjahrsputz bestens geeignet. Ihr volles Potential kann diese App aber wieder erst mit root ausspielen...



Auslagern

Genügt das noch immer nicht, stellt man sich oft die Frage: Kann man Apps eigentlich auf der SD-Karte installieren? Oder dorthin auslagern? Ab Android 2.2 (Froyo) ist dies zumindest teilweise bereits von Haus aus möglich. Die Funktionalität nennt sich naheliegenderweise "App2SD" – muss aber von der jeweiligen App explizit unterstützt werden (andernfalls lässt sich ein Verschieben zwar u. U. erzwingen, aber das kann zu unerwünschten Nebeneffekten führen). Ist dies der Fall, kann man sie aus dem Anwendungsmanager (*Einstellungen* → *Anwendungen* → *Anwendungen verwalten*) mit dem Button *Auf SD-Karte verschieben* auf die selbige befördern. Zumindest teilweise, denn ein gewisser "Grundstock" verbleibt dabei im internen Speicher. Unter anderem auch der zur App gehörende Dalvik-Cache.

Zwar steht anschließend wieder mehr interner Speicher zur Verfügung – die Sache hat jedoch einen kleinen Haken. Sobald man den Androiden per USB-Kabel an den heimischen Computer anschließt, und die Karte an selbigen "freigibt",

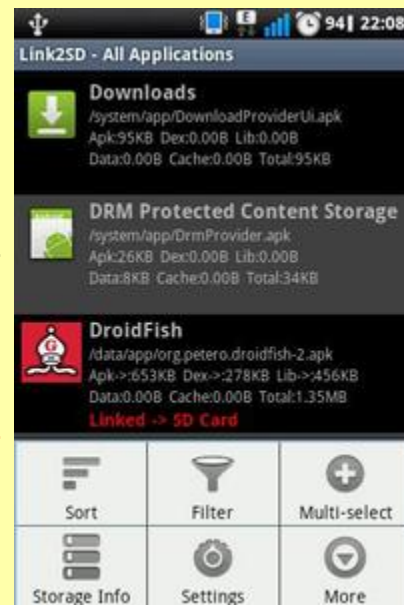
steht sie lokal ja nicht mehr zur Verfügung: Vom Androiden aus ist ein Zugriff erst dann wieder möglich, wenn die USB-Verbindung getrennt wurde. Die mit App2SD auf die Karte verschobenen Apps können bei am Computer eingebundener SD-Karte also nicht ausgeführt werden. Dies führt insbesondere zu Problemen, wenn Widgets involviert sind – weshalb man Apps mit Widgets besser nicht auf der Karte installiert.

Einige Custom-ROMs bieten daher App2SD+ an. Dies benötigt allerdings eine eigenständige (zusätzliche) Partition, die mit dem Dateisystem **Ext3** versehen sein muss. Wird nun eine USB-Verbindung zum PC hergestellt, verbleibt diese Ext3-Partition beim Androiden – freigegeben wird lediglich die "herkömmliche" **FAT** Partition. Somit stehen die ausgelagerten Apps weiterhin lokal zur Verfügung.

Ein weiterer Vorteil von App2SD+ ist, dass auch der Dalvik-Cache mit auf die Karte wandert: Es wird also deutlich mehr Platz freigeschaufelt. Als eingehendere Lektüre dazu empfiehlt sich [dieser Artikel in der Brutzelstube](#).

Wer noch auf einer Android-Version vor Froyo festsetzt, oder kein App2SD+ unterstützendes ROM verwendet, findet in **Link2SD** (rechtes Bild) eine Alternative. Auch diese Lösung benötigt eine separate Partition (es muss jedoch nicht zwingend Ext3 sein – FAT wird von *Link2SD* ebenfalls unterstützt), auf welche man sodann die Apps verschieben kann. Im Unterschied zu App2SD werden die Apps hier allerdings vollständig auf die Karte befördert, und im internen Speicher sodann durch einen passenden **symbolischen Link** ersetzt. Damit ist die Kompatibilität sichergestellt: Dem System wird so vorgegaukelt, dass sich die ausgelagerte App nach wie vor im internen Speicher befinden würde.

Wie bei App2SD+ gilt in diesem Falle: Über die USB-Verbindung wird die "App-Partition" einfach nicht freigegeben, sodass Apps und ihre Widgets weiterhin lokal verfügbar bleiben. Und der Dalvik-Cache lässt sich mit *Link2SD* ebenfalls auslagern, so man dies wünscht.



Need For Speed

Der Wunsch nach "mehr Speed" steht in der Regel dem Wunsch nach längeren Akku-Laufzeiten entgegen: Was mehr Leistung bringt, benötigt meist auch mehr Energie. Ausnahmen hiervon sind ein von unnötigen Altlasten befreiter Cache (siehe [Cache bereinigen](#)), die Deinstallation nicht mehr benötigter Apps einschließlich der Beseitigung etwaiger Rückstände (siehe [Aufräumen](#)), sowie genügend große Intervalle bei der Synchronisierung der Hintergrunddaten (siehe [Hintergrunddaten und Synchronisierung](#)) – diese Dinge eignen sich für beide Zwecke. Weiteren Punkten wollen wir uns im Folgenden widmen.

RAM bereinigen

Wer bei diesem Thema an Task-Killer denkt, liegt völlig daneben. Aber das erwähnte ich ja bereits bei den [generellen Maßnahmen](#). Task-Killer sind dafür da, Amok-laufende Apps zu beenden, deren man anderweitig nicht habhaft werden kann. Um die Speicher-Bereinigung kümmert sich Android eigentlich selbst recht gut. Zum Verständnis an dieser Stelle ein kurzer Ausflug hinter die entsprechenden Kulissen:

Der Grundgedanke ist, dass das Laden einer App aus dem internen Speicher bzw. gar von der SD-Karte naturgemäß (zusätzliche) Energie benötigt. Ist selbige hingegen bereits geladen, spart man sich den Ladevorgang. Also versucht Android, einmal geladene Apps im Speicher zu halten – auch, wenn der Anwender diese bereits wieder verlassen hat. Da nicht unbegrenzt RAM zur Verfügung steht, kommt dafür ein ausgeklügeltes System zum Einsatz.

Verantwortlich für das Aufräumen ist der sogenannte OOM-Killer (OOM steht für **O**ut **O**f **M**emory). Steht nicht mehr genügend freier Arbeitsspeicher zur Verfügung, ist der OOM-Killer dafür verantwortlich, solchen zu schaffen – indem er "irgend etwas" aus dem RAM schmeißt. Für die Auswahl dieses "Verlierers" werden verschiedene Kriterien verknüpft. Um diesen Vorgang verständlich zu halten, stelle ich das einmal vereinfacht dar:

Zunächst werden im RAM befindliche Prozesse in "Klassen" eingeteilt. Hierbei erhalten ständig benötigte System-Prozesse eine sehr hohe Priorität. Auf der nächsten Stufe stehen "sichtbare" Apps (also die, welche der Anwender gerade vor sich sieht, weil er damit arbeitet). Einige "Klassen" weiter folgen am Ende dann die "leeren" Apps – das sind die, die vom Benutzer "Beendet" wurden, und die keine Tätigkeiten mehr ausführen. Naheliegenderweise wird beim Aufräumen mit der niedrigsten Priorität angefangen. Für die "engere Auswahl" kommen dann noch weitere Kriterien ins Spiel, die wir an dieser Stelle nicht näher betrachten wollen.

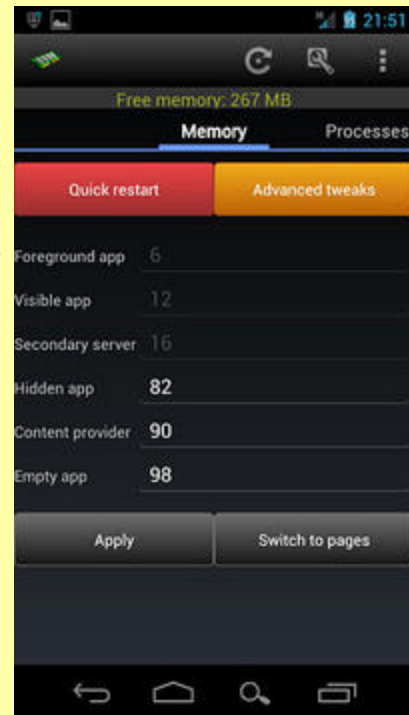
Woher weiß der OOM-Killer nun, wann er tätig werden soll? Dazu sind für jede "Klasse" Schwellwerte im System konfiguriert. Ein Bild sagt mehr als tausend Worte, und so erklären sich diese Schwellwerte am besten anhand des rechten Screenshots, der einen Bildschirm der App [AutoKiller Memory Optimizer](#): Je niedriger die Priorität, desto mehr freies RAM muss verfügbar sein, damit die App geladen bleibt.

Unschwer zu erraten: Diese App ist meine Empfehlung für diesen Zweck. Als Alternative käme [Auto Memory Manager](#) in Betracht. Beide Apps benötigen root-Rechte – da die genannten Schwellwerte im System verankert sind.

Natürlich gilt es nach Installation der jeweiligen App, die richtigen Einstellungen zu finden. Diese sind natürlich zum Einen stark vom Gerät (und dem dort insgesamt vorhandenen RAM) abhängig – ist weniger RAM verfügbar, fallen auch die Grenzwerte geringer aus. Zum Anderen ist da auch noch ein subjektiver Faktor, der nicht verleugnet werden kann. Zum Herantasten bieten beide genannten Apps gute "Presets"

(Voreinstellungen) an. Es empfiehlt sich, zunächst mit "Moderate" (so heißt das entsprechende Preset) zu beginnen, und sich von dort aus an die best geeigneten Werte heranzutasten.

Sind die richtigen Werte gefunden, und das System läuft damit flüssig und stabil? Dann weist man im letzten Schritt die ausgewählte App an, diese Werte nach jedem Neustart automatisch zu laden. Aber erst, wenn man sich dessen wirklich sicher ist! Bei zu aggressiven Einstellungen kann das andernfalls zu einem Dauer-Boot-Kreislauf führen, weil beispielsweise das System nach jedem Start gleich wieder "abgeschossen" wird. Um dies im Ernstfall zu vermeiden, lässt sich zumindest *AutoKiller Memory Optimizer* so konfigurieren, dass die Einstellungen um 2 Minuten verzögert geladen werden – damit hat man im Ernstfall noch die Möglichkeit, einzugreifen. Nebenbei bringt diese App noch etliche weitere System-Tweaks mit, die sich optional aktivieren lassen. Details dazu finden sich auf der [Projektsite](#).



Swap-Space nutzen

Unter Windows gibt es hierfür die "Auslagerungs-Datei", unter Linux richtet man sich am besten eine Swap-Partition ein oder greift, falls man dies bei der Installation vergessen hat und es schnell gehen muss, ebenfalls zu einer Swap-Datei. Was aber hat es mit diesem "Swappen" oder "Auslagern" auf sich?

Wird der Arbeitsspeicher (das RAM) knapp, muss Platz geschaffen werden. Im vorigen Abschnitt habe ich erläutert, in welcher Form sich der OOM-Killer um diese Angelegenheit kümmert. Wurde eine App auf diese Weise aus dem RAM

entfernt, müssen bei einem Neustart derselben sämtliche benötigte Speicher-Strukturen neu aufgebaut werden. Beim Swappen hingegen wird ein Abbild des entsprechenden Speicherbereiches auf den Datenträger ausgelagert – ein Neuladen geht auf diese Weise wesentlich schneller und energiesparender vonstatten.

Da böte sich doch eigentlich an, dass dies von Haus aus bereits eingerichtet ist – wo also ist der Haken? Am Datenträger hängt er. Während bei Desktop-Rechnern "normale Festplatten" zum Einsatz kommen, verwenden unsere Androiden fast ausnahmslos sogenannten Flash-Speicher. Und der ist hinsichtlich der Schreibzyklen ein wenig empfindlich: Zu viele davon verträgt er nicht. Wer also permanent auf dem gleichen Bereich schreibt (und das ist beim Swapping kaum auszuschließen), verringert damit die Lebensdauer des Datenträgers. Das ist für den Einen "schlimm" – der Andere hingegen "pfeift darauf", und kauft sich halt öfter einmal eine neue SD-Karte.



Da es sich hier wieder um einen tiefen Eingriff ins System handelt, bleibt auch diese Tuning-Maßnahme Anwendern mit root-Zugriff vorbehalten. Wer Swap nutzen möchte, dem sei die App [Swapper](#) (linkes Bild) ans Herz gelegt. Mit dieser lassen sich nicht nur alle notwendigen Einstellungen bequem von einer grafischen Oberfläche aus vornehmen – die App kümmert sich auch sonst um alles nötige. So verringert sie beispielsweise bei entsprechender Konfiguration das Risiko eines schnellen "Aufbrauchens der verfügbaren Schreibzyklen", indem sie die Swap-Datei bei jedem Neustart an anderer Stelle auf dem Datenträger unterbringt. Im Einzelnen kümmert sich *Swapper* um

- das Neuanlegen einer Swap-Datei beim Boot (an immer neuen Stellen auf dem Datenträger)
- ein sauberes Abschalten des "Swapping", wenn der Datenträger per USB gemountet wird
- die Neu-Aktivierung nach Lösen des USB-Mount
- gezieltes Einstellen der "Swap-Prioritäten"

sind nur die wichtigsten Punkte – und jeder davon ist optional. Es lässt sich auch eine Swap-Partition nutzen (das erübrigt dann auch das Abschalten bei USB-Kontakt, sowie das spätere wieder Aktivieren). In der Regel ist eine Swap-Partition ein wenig schneller als eine Datei – aber aufgrund der begrenzten Schreibzyklen von SD-Karten hat man länger etwas von selbigen, wenn Swapper sich darum kümmert, die Datei bei jedem Neustart an anderer Stelle der Karte wieder anzulegen.

Nebenbei: Übertreiben sollte man es auch nicht, ein gutes Mittelmaß ist gefragt. Ist zu viel "Krams" ausgelagert, bremst das am Ende doch wieder. Gute Anfangswerte sind 32..64MB Swap sowie eine "Swappiness" von 10..20.

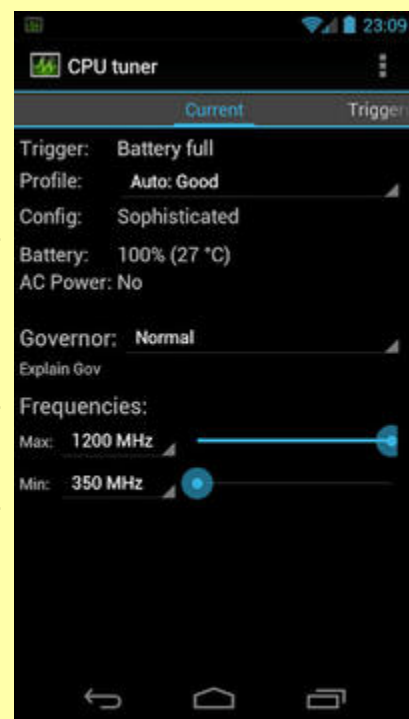
CPU übertakten

Um das Untertakten der CPU haben wir uns bereits in einem [früheren Abschnitt](#) gekümmert – jetzt soll es also in die andere Richtung gehen. Aber auch das geht natürlich nur mit root-Rechten.

Die verfügbaren Mittelchen (auch Apps genannt) sind natürlich die gleichen wie im genannten Abschnitt: [SetCPU](#) war ja bereits genannt, [CPU-Tuner](#) (rechtes Bild) ist eine gängige Alternative.

Da es beispielsweise wenig Sinn macht, den Androiden im Leerlauf mit "vollem Speed" laufen zu lassen (wenn man schläft, hat man daran in der Regel wenig Bedarf), bieten beide Apps auch "Timer" an – sowie die Möglichkeit, unterschiedliche Taktraten für aktiviertes und deaktiviertes Display zu verwenden. Alternativ lassen sich, wie bereits beschrieben, die CPU-Einstellungen auch sehr granular mit [Tasker](#) vornehmen: So kann man es beispielsweise einrichten, dass beim Start bestimmter ausgewählter Apps die Taktrate der CPU entsprechend angepasst – und bei Beendigung/Verlassen derselben der Standard wieder hergestellt wird. Dies garantiert ein gesundes Mittelmaß zwischen Performance und Akku-Laufzeit – ohne dabei mittelmäßig zu sein.

Vorsicht ist natürlich angesichts der "Obergrenze" (der höchstmöglichen Taktrate) geboten: Ab einem bestimmten Schwellwert wird das System instabil. Dann folgt der Kaffeewärmer, der Teekocher, und schließlich der Ziegelstein (Brick) – womit das Gerät dann nach Finnland kann. Dort wurde anscheinend die Sportart des [Handy-Weitwurfs](#) erfunden. Zumindest gibt es dort seit dem Jahr 2000 die entsprechende Weltmeisterschaft. Der Rekord liegt seit 2005 bei [gut 90 Metern](#).



ANHANG

Fragen und Antworten

Bei Woody Allen hieße dieses Kapitel sicher: *Was Sie schon immer zu Android wissen wollten – sich aber nie zu fragen trauten..* Nur keine Hemmungen! Immer her mit den [Fragen an Radio Eriwan](#)! Das Prinzip solcher Fragen erklärt der verlinkte Wikipedia-Artikel treffend mit einem kurzen Beispiel:

Anfrage an Radio Eriwan:

Stimmt es, dass Iwan Iwanowitsch in der Lotterie ein rotes Auto gewonnen hat?

Antwort:

Im Prinzip ja. Aber...

- es war nicht Iwan Iwanowitsch, sondern Pjotr Petrowitsch.
- und es war kein Auto, sondern ein Fahrrad.
- und er hat es nicht gewonnen, sondern es ist ihm gestohlen worden.

Alles andere stimmt.

Jeder wusste, was gemeint war – aber keiner konnte die (versteckte) Kritik am System wirklich "dingfest" machen. So war der politische Witz im Osten. Und auch wenn sich bei manchen vollmundigen Versprechen der Industriegrößen nicht selten ähnliche Fragen aufdrängen (gesagt wird A – aber jeder weiß, dass eigentlich B dahinter steckt), geht es im Folgenden hoffentlich nicht ganz so wild zu...

Apps & Co.

Was passiert bei Deinstallation einer App mit ihren Daten?

Ganz einfach: Sie werden gelöscht. Alles, was das System der zu deinstallierenden App explizit zuordnen kann, wird mit entfernt. Dazu gehört auch das [Verzeichnis](#), in dem die App ihre Konfigurationsdateien und ggf. Daten (im internen Speicher) abgelegt hat. Auch etliche andere Dateien ließen sich recht einfach zuordnen: Da jede App unter Android einen eigenen "User" darstellt, lässt sich schließlich ermitteln, welche Dateien ihr gehören.

Eine Ausnahme bildet dabei die SD-Karte: Hier erlaubt das Dateisystem keine Benutzerzuordnung. Daher lassen sich Dateien auf der SD-Karte nicht so leicht zuordnen, und bleiben demzufolge liegen.

Will Herr oder Frau Poweruser doch einmal eine App deinstallieren, und dabei die Daten behalten, lässt sich dies mit Hilfe des [SDK](#) erreichen:

```
pm uninstall -k com.app.wegdamit
```

deinstalliert die App mit dem Paketnamen `com.app.wegdamit` – behält (-k = "keep", behalten) aber die Daten auf dem Gerät.

Natürlich hätte man sie auch mit *Titanium Backup* sichern, und anschließend wieder herstellen können – *Titanium Backup* fragt ja auch brav: Die App, die Daten, oder beides?

Ist das Löschen von Cache und Daten das Gleiche wie Entfernen und Neu-Installation einer App?

Hin und wieder empfiehlt ein Entwickler bei speziellen Problemen mit seiner App diese zu deinstallieren und anschließend neu zu installieren. So manch einer fragt sich: Wäre da das Löschen von Cache und Daten nicht ausreichend? Ist das nicht dasselbe, vom Resultat her?

Ist es nicht. In beiden Fällen werden zwar App-Cache und Daten gelöscht. Bei einer Deinstallation wird aber auch der [Dalvik-Cache](#) der App entfernt, und bei einer Neuinstallation wieder frisch aufgebaut. Und manchmal kann genau an dieser Stelle "der Wurm" liegen.

Wo speichert Android die Metadaten zu meinen Fotos?

Äh: Was bitte sind Metadaten – vielleicht sollte das zuerst geklärt werden: Das sind etwa die Stichworte, die man den Fotos zuweisen kann. Und einiges anderes mehr: Welche Covers von Musikalben sich auf dem Gerät finden (und wo), welche Musikalben von welchen Künstlern, Audio-Genres, Playlisten, Videos... Irgendwo muss der Medien-Scanner das ja alles lassen!

Und wo wäre das? Da hierfür der sogenannte *Media Content Provider* zuständig ist, landen die Daten in dessen Datenbanken – die sich unter `/data/data/com.android.providers.media/databases` finden. Und zwar fein getrennt in mindestens zwei Datenbanken: `internal.db` für den internen Speicher, und `external-<Geräte-ID>.db` für die jeweilige Speicherkarte. Herankommen tut an diese Dateien natürlich wieder einmal nur Freund root – und bei Forensikern erfreuen sie sich besonderer Beliebtheit...

Wer genaueres zu den Strukturen der in den genannten Datenbanken enthaltenen Tabellen wissen möchte, findet beispielsweise in der [Entwickler-Doku](#) weitere Angaben – in englischer Sprache, versteht sich...

Wie kann ich eine App im Speicher halten?

Ich benutze App xyz ständig. Doch fast immer, wenn ich zu ihr wechseln möchte, wird sie neu gestartet. Kann ich Android irgendwie dazu bringen, sie im Speicher zu halten?

Diese Frage haben sich die meisten schon einmal gestellt: Warum schließt Android laufend unsere Lieblings-App? Das Prinzip habe ich ja bereits beim Thema [RAM Bereinigen](#) erklärt: Wird der Speicher knapp, schlägt der [OOM-Killer](#) zu. Mit welcher Wahrscheinlichkeit ihm nun unsere Lieblings-App zum Opfer fällt, hängt unter anderem davon ab, zu welcher "OOM-Klasse" sie gehört. Ändern kann man diese wieder einmal nur mit root – und deshalb male ich gleich wieder meinen Kasten um das Folgende:

[CyanogenMod](#) bietet zumindest für den Standard-Launcher eine entsprechende Option – aber auch nur für diesen. Dafür kann die App einfach in eine "wichtigere Klasse" eingeteilt werden, indem ihr oom_adj-Wert gesenkt wird. Wie man dies für eine beliebige App erledigen kann, erklärt [ein Post bei XDA](#): Man erstellt eine Datei namens etc/init.d/99applock, und füllt sie mit folgendem Inhalt:

```
#!/system/bin/sh

sleep 60

PPID=$(pidof com.your.app)
echo "-17" < /proc/$PPID/oom_adj
renice -18 $PPID
```

com.your.app ist dabei mit dem Paketnamen der gewünschten App zu versehen – und etc/init.d/99applock mit den passenden Rechten: chmod 777 etc/init.d/99applock.

Unterstützen alle Android-Apps NFC?

Das brauchen sie gar nicht – darum sollte sich das System selbst kümmern. Lediglich passende "Intents" müssen sie bereitstellen, über die sie von außen angesprochen und mit entsprechenden Daten gefüttert werden können (vergleichbar mit [APIs](#)). Um das Auslesen und Interpretieren der NFC-Tags kümmert sich der zentrale NFC-Service, der dann auch die passende Aktion auslösen sollte.

Vergleichen lässt sich das am besten mit den Barcode-Readern, die ja ebenfalls die passende Aktion wählen: So muss etwa die Kontaktverwaltung auch keine Barcode-Unterstützung haben. Erkennt der Barcode-Reader eine Adresse, ruft er die Kontaktverwaltung auf, und übergibt ihm diese. Oder die Telefonnummer an die Telefon-App, bzw. die URL an den Web-Browser.

Passende NFC-Apps wären etwa [on{X}](#), oder [NFC Launcher](#). Und natürlich muss auch das Gerät selbst NFC unterstützen.

Ich habe meinen einzigen Launcher gelöscht!

Die Tücken von root und Custom ROMs: Einem "normalen Anwender" dürfte das nicht passieren, da der "Standard Launcher" im read-only Bereich des Systems installiert ist. Ohne Launcher sieht es unter Android natürlich mau aus: Eingehende Anrufe lassen sich zwar noch annehmen, aber man kann keine Apps mehr starten. Wenn das Kind nun in den Brunnen gefallen ist, gibt es dennoch eine ganz triviale Lösung:

Einfach am PC mit dem Web-Browser den [Google Playstore](#) besuchen, sich mit dem auf dem Gerät verwendeten Google-Konto anmelden, sich einen alternativen Launcher aussuchen, und die Installation von dort aus anstoßen. Kurz darauf ist wieder ein Launcher verfügbar, und der Androide somit voll einsatzbereit.

Backup & Co.

Hilfe! Ich habe versehentlich etliche (einen ganzen Thread) SMS/MMS gelöscht!

Das passiert leicht: Ein unachtsamer Tappser – und statt einer einzelnen SMS verschwindet gleich der ganze Thread im Daten-Nirvana. Sofern kein aktuelles [Backup der SMS-Daten](#) existiert, sieht es da schlecht aus (und würde ein solches existieren, würde man sicher nicht nach einer Antwort suchen müssen). Gibt es da noch Chancen?

Mit Android < 4.0: Nur für root, der über das [Custom-Recovery](#) sofort ein [Nandroid-Backup](#) anlegt. Mit Android ab Version 4.0 kann man den Schritten unter [Komplett-Backups ohne root](#) folgen – und hoffen, dass dies die nötigen Dateien enthält.

Diese finden sich nach dem Entpacken der Image-Datei der Datenpartition im Verzeichnis `data/data/com.android.providers.telephony/databases/` (ja, root könnte sich dieses auch mit einem Dateimanager oder per `adb pull` besorgen). Mit einem Programm wie etwa [Sqliteman](#) lässt sich nun etwa die im genannten Verzeichnis befindliche Datei `mmssms.db` nach den gelöschten Nachrichten durchsuchen. Auch die Datei `mmssms.db-wal` (eine SQLite Journal-Datei – genauer gesagt: Das [Write-Ahead-Log](#)) enthält häufig weitere diesbezügliche Daten. Vor dem Bearbeiten mit irgendwelchen Tools legt man sich davon am besten Kopien an – damit man bei einem Fehlschlag nicht wieder ganz vorn anfangen muss.

Ach ja: Genauso funktioniert es natürlich auch mit jeder anderen Datenbank, die beliebige andere App in ihrem Datenverzeichnis abgelegt hat.

Die SMS sind noch da – dafür habe ich aber eine Reihe Dateien gelöscht!

Auch kein Weltuntergang. Sicher waren diese auf der SD-Karte – nicht-root Anwender also bitte einfach nach dem Kasten weiterlesen...

Die Dateien waren im internen Speicher? Aha, mit dem root-Explorer gespielt. Dann sollte schnellstens eine Sicherung der Partition durchgeführt werden. Anbieten tun sich hier [Nandroid](#), aber auch `dd`. Da letzteres sich für eines der unter dem Kasten aufgeführten Recovery-Tools besonders gut eignet, dazu ein wenig mehr.

Zuerst muss herausgefunden werden, welche Partition denn benötigt wird. Ist dies nicht bekannt, kann dazu (per [ADB](#) oder mit einer Terminal-App) der Befehl `mount` ausgeführt werden. Ohne Parameter aufgerufen, zeigt dieser nämlich alle eingebundenen Dateisysteme an. Ich kürze diese Ausgabe hier einmal ein wenig ab:

```
$ mount
<snip>
/dev/block/mmcblk1p26 on /data type ext3 (rw,nosuid,nodev,noatime,nodiratime,errors=continue,data=ord
/dev/block/vold/179:1 /mnt/sdcard vfat <snip>
```

Befanden sich die gelöschten Daten also beispielsweise unterhalb von `/data/`, muss sich hier das Gerät `/dev/block/mtdblock6` gemerkt werden. Des

weiteren ist auch zu sehen, dass sich die SD-Karte unter /mnt/sdcard finden lässt – und hoffentlich genügend Platz aufweist:

```
$ df -h /mnt/sdcard /data/data
Filesystem      Size      Used Available Use% Mounted on
/dev/block/vold/179:1  14.9G    1.7G    13.2G   11% /mnt/sdcard
/dev/block/mmcblk1p26  6.5G    382.1M    6.1G    6% /data
```

Über 13 GB frei auf der Karte, knapp 7 GB maximal werden benötigt: Passt. Es kann also zur Tat geschritten werden. Bei Aufruf des Befehls dd genau auf die Parameter achten! Nicht umsonst wird gemunkelt, "dd" stünde für "disk destroyer": Verwechselt man hier die Eingabe und Ausgabe, ist letztere danach futsch. Also:

```
dd if=/dev/block/mmcblk1p26 of=/mnt/sdcard/data.img
```

Kurz erklärt: "Disk Duplicator, lese von /dev/block/mmcblk1p26 und schreibe alles, was dort ist, nach /mnt/sdcard/data.dd." "if" steht hier also für Eingabe-Datei (englisch "input file"), "of" für "Ausgabe-Datei" ("output file"). Letzteres ist in unserem Beispiel tatsächlich eine Datei – die Eingabe jedoch ein Gerät. Doch unter Unix/Linux sind auch Geräte Dateien, siehe Wikipedia: [Gerätedatei...](#)

Zum Retten der Daten wird nun die SD-Karte an den PC angeschlossen, und ein zuvor heruntergeladenes Datenrettungs-Programm darauf losgelassen. Davon gibt es eine ganze Reihe, und drei möchte ich hier auch benennen:

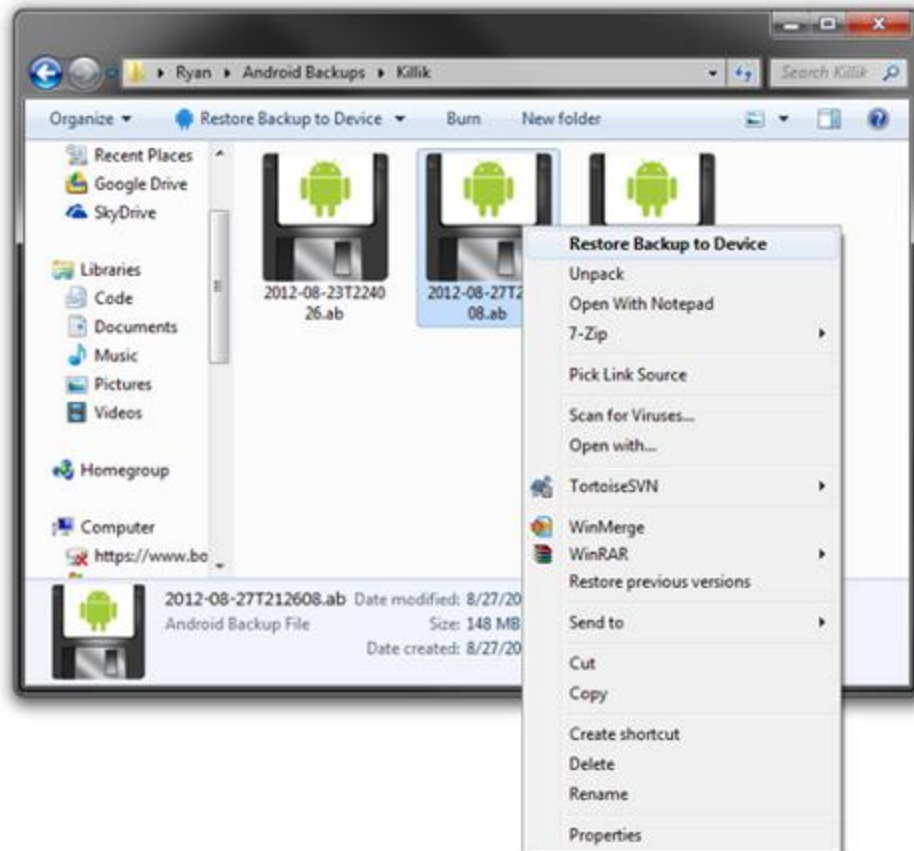
- [Scalpel](#) gibt es für Linux (bevorzugt), Mac OS X und Windows. Es ist eines der mächtigeren Tools in dieser Klasse – erfordert aber auch ein wenig Eingewöhnung.
- [foremost](#) gibt es (nur?) für Linux. Da es aber auch mit dd Images umgehen kann, ist dies für den im obigen Kasten beschriebenen Fall die passende Lösung.
- [TestDisk](#) unterstützt die größte Anzahl an Betriebssystemen, und stellt auch eine menügeführte Benutzeroberfläche bereit.

Details zur Nutzung finden sich auf den verlinkten Webseiten des jeweiligen Tools – und würden an dieser Stelle den Umfang des Kapitels sprengen...

Kann ich Daten aus einem verschlüsselten ADB-Backup extrahieren?

Unter [Komplett-Backups ohne root](#) habe ich ja beschrieben, wie man ab Android 4.0 auch als "Normalo" vollständige Backups erstellen kann – Apps und auch ihre Daten. Dabei musste ein Passwort angegeben werden, mit dem das Backup verschlüsselt wird – gegen unbefugte Zugriffe. Die Zugriffe des Eigentümers sind sicher nicht unbefugt (zumindest, wenn es um die Daten geht) – aber wie kommt er da nun heran, ohne sie auf dem Androiden wieder herzustellen (und damit aktuellere Daten zu überschreiben)?

Der bereits bei besagtem Backup genannte [Ryan Conrad](#) hat natürlich auch dafür [eine Antwort](#) parat. Er hat sich eingehend mit dem Thema beschäftigt, da er schließlich dieses Backup/Restore in seinem Programm implementierte. Die einfachste Antwort lautet also: Sein Programm [Droid Explorer](#) in Version 0.8.8.7 oder höher zu installieren, und die gewünschte Datei damit entpacken:



Dieser Tipp gilt zumindest derzeit nur für diejenigen, die Windows auf ihrem Computer einsetzen. Für alle anderen ist ein mögliches Vorgehen auf der verlinkten Seite erklärt. Um wenigstens einen kleinen Anhaltspunkt zu nennen: Die .ab Datei ist ein mit AES verschlüsseltes Archiv, in dem sich eine [TAR](#)-Datei befindet. Und es gibt ein Tool namens [Android Backup Extractor](#) (kurz: ABE), mit welchem man diesen Tarball aus der .ab Datei herausholen kann:

```
java -jar abe.jar unpack <backup.ab> <backup.tar> <password>
```

Da es sich bei *ABE* also offensichtlich um ein Java-Programm handelt, sollte es auf den meisten Betriebssystemen laufen.

Panik! Nandroid findet kein sd-ext!

Da erstellt man also pflichtbewusst sein [Nandroid Backup](#) - und die Bildschirmausgabe endet mit

```
Backing up system...
Backing up data...
Backing up .android_secure
Backing up cache
No sd-ext found. Skipping backup of sd-ext.
Generating md5 sum...

Backup complete!
```


Was heißt hier: No sd-ext found? Habe ich jetzt ein Problem? Ist da etwas "futsch"?

Mit Neffen und mitnichten. Das bedeutet lediglich, dass die SD-Karte nicht über eine EXT3 bzw. EXT4 Partition verfügt, die auf diesen Namen hört. Eine solche wurde/wird von verschiedenen [Custom ROMs](#) für ein erweitertes *Apps2SD* verwendet. Gibt es diese Partition also nicht, kommt diese Art von *App2SD* auf dem Gerät wahrscheinlich auch nicht zum Einsatz – die Meldung ist also kein Grund zur Sorge. Andernfalls würde sie nachdrücklicher zum Ausdruck gebracht, und nicht nur "beiläufig ausgegeben" werden.

Welche Daten sichert eigentlich Google Backup?

Bei der Ersteinrichtung wird man (zumindest ab Gingerbread) ja gefragt, ob man seine Daten bei Google sichern möchte. Was aber wird dabei eigentlich gesichert, wenn man die Frage bejaht?

Ganz offensichtlich sind damit nicht die Kontakte oder Kalenderdaten gemeint – denn die werden auch in die Google-Cloud synchronisiert, wenn man diese Frage verneint. Im [Honeycomb User Guide](#) (englisches PDF) heißt es dazu sinngemäß:

- Android Settings, wie WLAN Netzwerke und Passworte, Benutzerwörterbuch, usw.
- Die Einstellungen vieler Google-Apps, wie etwa Browser-Lesezeichen
- Die aus dem Playstore heruntergeladenen Apps

Zumindest der letzte Punkt macht wenig Sinn, da diese Informationen ja ohnehin im Playstore hinterlegt sind. Weiter heißt es jedoch, dass auch einige Apps von Drittanbietern diese Backup-Schnittstelle nutzen, um die Daten ihrer Apps zu sichern. Einige, nicht alle.

Wenn man sich nun mit einem neuen bzw. auf Werkseinstellungen zurückgesetzten Gerät erstmalig mit seinem Google-Konto wieder anmeldet, sollen diese Daten wieder hergestellt werden.

Soweit die Theorie. In der Praxis schlägt leider die Wiederherstellung oftmals fehl, wie zahlreiche Berichte zeigen – man sollte sich also keinesfalls auf diese Funktionalität verlassen. Zumal der Umfang der gesicherten Daten ohnehin recht eingeschränkt ist.

Rund um die SD-Karte

Warum kann ich App xyz nicht auf die SD-Karte verschieben?

Das kann verschiedene Ursachen haben:

- Die App unterstützt kein *App2SD* (hat der Entwickler so festgelegt). In diesem Fall wird die Option zum Verschieben erst gar nicht angeboten.
- Die App war dort zuvor bereits installiert (und man hat beispielsweise einen Werksreset gemacht).

Während im ersteren Falle der Entwickler Abhilfe schaffen kann (es sei denn, die App beinhaltet Widgets oder Hintergrund-Dienste – dann darf sie nicht auf die SD-

Karte), gilt es in letzterem Falle, selbst Hand anzulegen. Hier sind wahrscheinlich ein paar "Dateileichen" auf der Karte liegen geblieben – und die müssen weg. Am einfachsten geht das natürlich, wenn man die SD-Karte mit einem Kartenleser an einen Computer anschließt, da Android (ohne root) das entsprechende Verzeichnis "versteckt" und keinen Zugriff zulässt. Das Vorgehen ist dann etwa folgendermaßen:

1. Den Paketnamen der entsprechenden App ermitteln. Am Einfachsten geht dies, indem man mit dem Web-Browser die App-Seite im Playstore aufsucht, denn dann steht der Paketname in der URL: Unmittelbar hinter dem id= (und endet, sobald ein & oder # auftaucht). Heißt die URL beispielsweise `https://play.google.com/store/apps/details?id=com.rovio.angrybirds&feature=search_result`, so lautet der Paketname `com.rovio.angrybirds`
2. Auf der SD-Karte nach der zugehörigen Datei suchen. Dies befindet sich unter `.android_secure`, und hieße in unserem Beispiel höchstwahrscheinlich `com.rovio.angrybirds-1.asec`.
3. Diese Datei löschen.
4. App auf SD verschieben – jetzt sollte es wieder klappen.

Ausführliche Gründe, die eine App für die Installation auf der SD-Karte disqualifizieren, nennt die [Android Entwickler-Seite](#). In der Regel hängt dies damit zusammen, dass die SD-Karte z. B. bei Bereitstellung an einem Computer auf dem Android-Gerät nicht zur Verfügung steht:

- **Services:** Eine App, die Hintergrunddienste bereitstellt. Diese Dienste stünden dann ggf. nicht zur Verfügung, und würden auch nicht automatisch neu gestartet.
- **Alarm Services:** Registrierte Alarmer würden bei Nichtverfügbarkeit der SD-Karte abgebrochen.
- **Live Wallpapers:** Könnten ohne die Karte nicht weiterlaufen, und würden durch das "Standard Wallpaper" ersetzt.
- **App Widgets:** Könnten nicht mehr aktualisiert werden, und "verschwinden" vom Homescreen. Meist sind sie erst nach einem Reboot wieder verfügbar.
- **Account Managers:** Können bei nicht verfügbarer SD-Karte natürlich auch nicht arbeiten, sind sie auf selbiger installiert. Die damit verbundenen Konten sind daher nicht sichtbar, wenn die Karte "nicht sichtbar" ist.
- **Sync Adapters:** Funktionieren nicht, wenn die Karte nicht verfügbar ist.
- **Device Administrators:** Wären dann ebenfalls außer Funktion, was die Geräte-Funktionalität negativ beeinflussen kann.
- **Broadcast Receivers, die auf *boot_completed* lauschen:** Da dieser Event (*boot_completed*) vor dem Einbinden der Karte gesendet wird, bekommen die auf der Karte installierten Apps davon nichts mit (hier geht es um "nach dem Booten Starten").
- **Copy Protection:** Wird Googles Kopierschutz genutzt, kann die betreffende App nicht auf der Karte installiert werden. Googles "Lizenz-Service" ist davon jedoch nicht betroffen.

Warum kann ich die App xyz nicht updaten?

Sie ist auf der SD-Karte installiert? Ein ähnliches Problem wie das vorige. Aber mit einer anderen Lösung – vorausgesetzt, es ist genügend interner Speicher frei:

1. App in den internen Speicher verschieben
2. Updaten (sollte jetzt funktionieren)
3. App bei Bedarf wieder auf die Karte verschieben

Klingt blöd – klappt aber so.

Muss ich beim Kauf einer SD-Karte auf die "Klasse" achten?

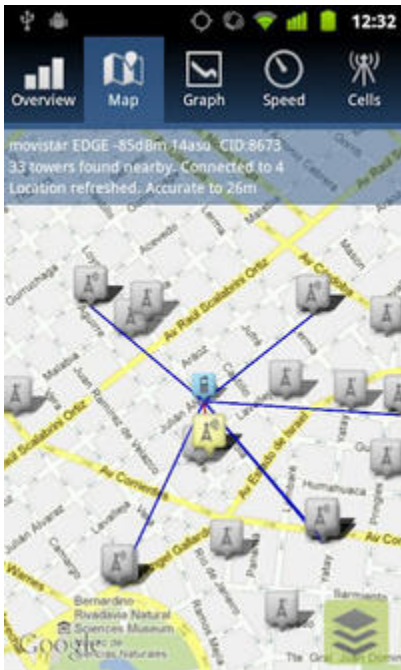
Das sollte man schon – besonders wenn man auf Performance und Akku-Laufzeit Wert legt. Wer heute eine SD-Karte kauft, sollte nicht aus Geiz-ist-Geil Gründen (Preis) zu einer "Class 2" Karte greifen – daran hat man wirklich keine Freude. Sie ist recht langsam (spätestens bei Video-Aufnahmen wird man das deutlich spüren), und im Vergleich mit "höheren Klassen" auch ein wenig Akku-hungrig.

Eine "Class 4" Karte dagegen ließe sich bereits als "Budget-Variante" einstufen, die für viele Einsatzbereiche ausreichend ist – bei Aufnahme von HD-Video könnte es aber auch hier eng werden. Generell empfiehlt es sich daher, noch den einen Euro darauf zu legen, und zumindest zu einer "Class 6" Karte zu greifen. Deren Spezifikationen sollten auch einer HD-Aufnahme Genüge tun.

Schaden tut eine höhere Klasse natürlich nicht – mehr Speed und schonenderer Umgang mit den Ressourcen haben schließlich noch niemanden gestört. Allenfalls der Preis: Je höher die Klasse, desto höherer auch dieser. Doch da liegt die Toleranzgrenze bei jedem anders.

Tuning

Was ist eigentlich "Mobilfunk-Standby", und warum braucht es so viel Akku?



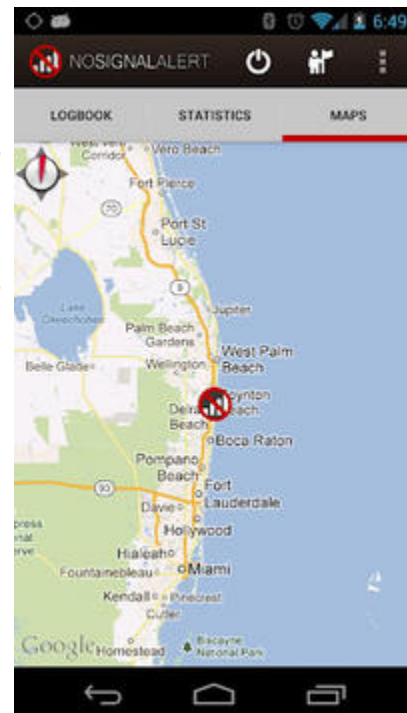
Hierbei handelt es sich nicht etwa um die "Hintergrunddaten im Standby", wie ein [Post bei den XDA-Developers](#) suggeriert. Vielmehr hat dieser (auf Englisch, "Cell Standby" bezeichnete) Zustand direkt etwas mit dem Mobilfunk-Signal zu tun. Oder mit der Abwesenheit desselbigen.

Was passiert, wenn das Signal zu schwach wird? Das Telefon sucht ein neues, stärkeres Signal. Im Regelfall ist ja auch mehr als nur ein Funkmast in der Nähe. Also wird der Antenne mehr und mehr Leistung zugeführt – in der Hoffnung, eine starke und stabile Verbindung gewährleisten zu können. Schließlich will ein Telefon telefonieren, und dazu benötigt es eine Verbindung. Da das System leider nicht intelligent genug zu sein scheint, nach einer angemessenen Zeit dabei eine Pause einzulegen, verbraucht es eben ggf. recht lange Zeit recht viel Strom.

Berichten zufolge (etwa in diesem [AndroidForums Post](#)) fällt dieser zusätzliche Verbrauch bei 4G deutlich höher aus als bei 3G. Und wirft man einen Blick auf die Übersicht der [Akkufresser-Daten](#) am Ende dieses Buches, wird klar: 3G frisst sicher noch immer mehr als 2G. Hat man alles gleichzeitig aktiviert, erfolgt die Suche dann auch in allen Bändern (also 2G, 3G und 4G). Daher kann es durchaus Sinn machen, sich auf einen Standard festzulegen, wenn man in einem Gebiet mit schlechter Abdeckung unterwegs ist.

Gänzlich vermeiden kann man dieses Problem natürlich, indem man seinen Androiden ausschließlich im Flugzeug-Modus betreibt – was selbstverständlich keine Lösung ist. Einige interessante Apps gibt es jedoch, die in diesem Zusammenhang genannt werden sollten:

- [No Signal Alert](#) protokolliert im Hintergrund, wo man sich in "toten Zonen" bewegt hat – und kann diese auf einer Karte anzeigen (rechtes Bild). Natürlich lässt sich auch die Log-Datei einsehen. Somit weiß man zumindest, wo dieses Unheil droht.
- Gleiches weiß auch [OpenSignalMaps](#) (linkes Bild) zu berichten. Zusätzlicher Nutzen dieser App: Sie zeigt auch an, welche Sendemasten sich in der Nähe befinden. Außerdem bietet sie einen "Kompass", der die



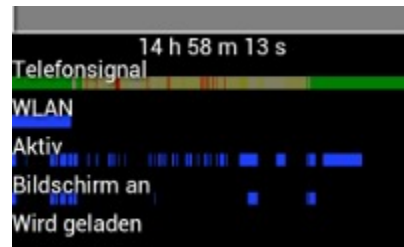
Richtung zur nächstgelegenen starken Zelle anzeigt. Versteht man also seinen Gesprächspartner immer schlechter, kann man sich Dank dieser App in die richtige Richtung bewegen, um den Empfang zu verbessern. Auch ein Widget ist hier an Bord. Beide Apps warnen auf Wunsch auch, wenn man eine "tote Zone" betritt.

- Den Empfang verbessern möchte hingegen [Network Booster](#). Diese App führt einfach einen "Reset" des Radios durch, wodurch die Netzverbindung frisch aufgebaut wird. Das kann natürlich nur dann etwas bringen, wenn auch (stärkere) Sender in Reichweite sind.
- Weitere [kleine Helferlein](#) wurden ja bereits benannt.

Ob man davon betroffen ist, lässt sich leicht an den Akku-Statistiken herausfinden (Screenshots rechts). Diese sind in den Systemeinstellungen enthalten, meist unter *Telefoninfo*. Taucht auf der ersten Seite *Mobilfunk-Standby* gleich bei den größten Verbrauchern auf, ist die Wahrscheinlichkeit hoch. Dann einfach die kleine Grafik oberhalb der Werte antippen, was zum zweiten Screenshot führt.



Hier ist der mit *Telefonsignal* beschriftete Balken genauer zu betrachten: Ein kräftiges Grün steht für guten Empfang – was der Verbrauchs-Graph darüber mit einer relativ flachen Kurve als "recht sparsam" ausweist. Geht der Farbton hingegen ins Gelbliche (mäßiger bis schlechter Empfang), oder wird gar rot (kein Signal), bestätigt auch der Graph darüber einen höheren Stromverbrauch: Die Kurve fällt stärker. Dass dies nicht etwa daran liegt, dass der Anwender gerade viel mit dem Gerät gemacht hat, lässt sich auch erkennen: Der Bildschirm war zu diesen Zeitpunkten (gelblicher bzw. roter Balken) meist ausgeschaltet (keine blaue Markierung bei *Bildschirm an*).



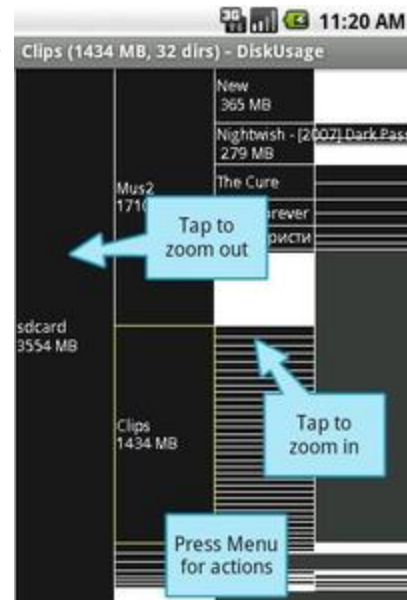
Ein [Youtube-Video zu Cell Standby](#) kann in diesem Zusammenhang ebenfalls recht aufschlussreich sein.

Wie man diesem Problem zu Leibe rücken kann, beschreibt auch das Kapitel [Akku-Verbrauch im Mobilfunk-Standby mit Tasker bekämpfen](#), welches sich etwas weiter hinten in diesem Buch findet. Neben Tasker werden dabei auch ein paar Alternativen benannt.

Wer frisst meinen Speicherplatz?

Egal, wie groß der Datenspeicher ist – es ist nie genügend davon vorhanden, er ist immer voll. Aber wo ist er abgeblieben? Und wie lässt sich das herausfinden?

- Das Menü *Anwendungen* → *Anwendungen verwalten* zeigt zum Einen eine "Gesamtzahl" für belegten und freien Speicher (auf volle Megabytes gerundet). Zum Anderen lassen sich hier die Anwendungen auch nach Größe sortieren, sodass sich besonders speicherhungrige Kandidaten aufspüren lassen.
- Auch wenn eine App mit *App2SD* auf die SD-Karte ausgelagert wurde, bleiben noch immer Reste im internen Speicher zurück (ein vollständiges Auslagern ist also nicht möglich).
- Temporäre Dateien und Cache belegen Speicherplatz. Letzterer lässt sich problemlos aufräumen (siehe [Cache Bereinigen](#)) – ersteres ist ein wenig problematischer (siehe unten). Zu diesem gehört übrigens auch das Verzeichnis, in dem der Playstore heruntergeladene Dateien vor der Installation zwischenspeichert. So kann es vorkommen, dass dabei die Fehlermeldung über "ungenügenden Speicher" erscheint – obwohl am Zielort noch genügend zur Verfügung steht: Es ist dann lediglich der "temporäre Speicher" voll.
- *Tombstones* (so nennen sich bei Android die unter Unix/Linux als [Coredump](#) bekannten Speicherauszüge, die bei Absturz eines Prozesses angelegt werden können) und [System-Log-Dateien](#) belegen ebenfalls Speicherplatz



Kleine Helferlein gibt es natürlich auch hier wieder:

- [Sandisk Memory Zone](#) zeigt die Kapazitäten (frei/belegt) der verfügbaren Datenspeicher (auch in der Cloud) an, und gibt ebenfalls Einblick in Details
- [DiskUsage](#) (rechtes Bild) erlaubt das Durchwandern der Verzeichnisse nach Dateigröße, womit sich die großen Verbraucher schnell aufspüren lassen sollten

Anwender mit root-Rechten haben natürlich wieder einmal weitere Möglichkeiten. Sie können direkt in der Terminal-App (oder via `adb shell`) auf den internen Speicher zugreifen. Um beispielsweise die fünf "größten Brocken" auf der Daten-Partition aufzuspüren, würde sich etwa folgender Befehl eignen: `su -c "du /data" | sort -rn | head -n 5`.

Auf gleiche Weise lassen sich natürlich auch die übrigen Dateisysteme erkunden. Welche es da gibt, verrät u. a. ein `df -h` – welches nebenbei auch noch die aktuellen Größenordnungen (freier und belegter Speicher) in gut lesbarem Format anzeigt.

Verschiedenes

Warum werden neue Medien nicht angezeigt?

Ich habe gerade eine ganze Reihe (Bilder / Videos / MP3-Dateien) per WLAN auf mein Android-Gerät kopiert. Warum zeigt (die Galerie / die Video-App / die Musik-App) nichts davon an?

Damit die genannten Apps nicht bei jedem Aufruf erst alle Verzeichnisse nach Medien durchsuchen müssen, hat man sich bei Android etwas Schlaues einfallen lassen: Den *Medien-Scanner*. Dieser wird bei bestimmten Ereignissen automatisch vom System aufgerufen, und durchsucht sodann den gesamten Speicher. Was er dabei findet, wird in eine Datenbank eingetragen. Somit können Apps wie die Galerie oder Video-/Musik-Apps einfach auf diese Datenbank zurückgreifen. Das geht wesentlich schneller – und spart obendrein auch noch Akku.

Der Haken ist: Bei welchen Ereignissen tritt der *Medien-Scanner* denn in Aktion? Die Antwort: Nach dem Systemstart – und nach dem Einbinden der SD-Karte. Punkt. Nicht nach dem Kopieren über das WLAN.

Abhilfen mit Bordmitteln wären hier:

- Das Gerät neu starten. Kein toller Vorschlag – klappt aber.
- Die SD-Karte kurz aus- und wieder Einhängen. Geht über die Systemeinstellungen: *Speicher* → *Speicherkarte entnehmen* – ist aber auch umständlich.
- Abwarten, bis der *Medien-Scanner* aus einem anderen Grund anspringt – dauert viel zu lange.

Eine einfache Abhilfe findet sich natürlich wieder einmal im Playstore, in Form von kleinen Apps wie [Rescan SDCard!](#). Diese kann man als Shortcut auf dem Homescreen ablegen, und so mit einem einfachen Antippen starten. Die App macht dann nichts anderes, als dem *Medien-Scanner* Bescheid zu geben: Aufwachen, es gibt etwas zu tun! Sodass dieser sofort in Aktion tritt.

Begriffserklärungen

ADB

Die **A**ndroid **D**ebug **B**ridge ist Bestandteil des Android-**SDK**. Anders als der Name es nahelegt, ist ADB für mehr als nur das **Debuggen** gut. So lässt sich hiermit ein Android-Gerät steuern und kontrollieren, Dateien können übertragen, installiert oder auch gelöscht werden, und mehr. Eine genauere Beschreibung findet sich auf der [Entwicklerseite](#).

AES

AES ist der Nachfolger des Verschlüsselungs-Standards **DES** – und bringt als solcher natürlich eine vergleichbar höhere Sicherheit mit.

AOSP

Hinter dieser Abkürzung verbirgt sich das **A**ndroid **O**pen **S**ource **P**roject – also die Kern-Entwicklung des Android-Systems, auf der alle weiteren Android-Entwicklungen (inklusive der Hersteller-spezifischen "Dreingaben" wie TouchWiz, MotoBlur, Sense, & Co. – aber auch die meisten **CustomROMs**) aufbauen.

API

Kurzform (oft gesprochen, wie man es schreibt – aber auch als Abkürzung buchstabiert) steht für **A**pplication **P**rogrammers **I**nterface. Gemeint ist hier eine definierte Schnittstelle, über die Informationen bezogen werden können. Die Android-API bietet auf diese Weise z. B. Informationen über Akkustand u. a.m.. So muss nicht jeder Programmierer das Rad neu erfinden.

APK-Datei

Die Abkürzung steht für **A**ndroid **P**ackage, und da drin befindet sich in der Regel eine App zur Installation unter Android. Da diese Apps ja in Java geschrieben sind, verwundert es sicher nicht, dass das APK Format eine "Abwandlung" des JAR (**J**ava **A**rchive) ist, und sich somit mittels WinZip & Co. ein Blick in selbige werfen lässt...

Datei-Manager unter Android erkennen diese Packages natürlich, und bieten an, die enthaltene App zu installieren.

APN

Kürzel für **A**ccess **P**oint **N**ame (zu Deutsch: Name des Zugangspunktes). Gemeint ist in der Praxis mitnichten nur der Name, sondern vielmehr der komplette Datensatz. Siehe auch mobiles Datennetz für eine detailliertere Beschreibung, sowie APNs für eine nach Netzanbietern sortierte Liste von APN-Definitionen.

App

Kurzform für *Application*. Wird auch im Deutschen ("**Neudeutsch**": Applikation) verwendet, da "Anw" einfach blöd klingt. Denn nichts anderes bedeutet das englische Wort *Application*: Anwendung.

Im Zusammenhang mit Smartphones aller "Couleur" (also Früchte wie auch KGMs, kleine grüne Männchen) hat sich die Kurzform "App" eingebürgert – "Application" (oder im Deutschen "Anwendung") wird hier eher selten verwendet.

Baseband

Oh, oder auch "Radio-ROM" bzw. "Radio-Image" wird es gern genannt. Das ist quasi die eigentliche Geräte-Firmware. Hat weniger direkt mit Android, als vielmehr mit der Hardware zu tun – und initialisiert letztere, so dass sie von ersterem genutzt werden kann. Also so etwas ähnliches wie das BIOS beim PC. Und genau wie dieses, befindet es sich i.d.R. auf einem separaten Chip.

Das Teil bootet also die Hardware, und übergibt dann an den eigentlichen Bootloader, der sich dann um Android kümmert.

Bloatware

Vom Hersteller oder Provider vorinstallierte Software, die nicht jedermann benötigt (beispielsweise die Aktien-App). Der Begriff leitet sich vom englischen Wort "to bloat", "aufblähen", ab – da diese Apps das ROM unnötig "aufblähen". Aufgrund der Tatsache, dass sie im ROM integriert sind, kann ein Anwender sie ohne root-Zugriff nicht entfernen.

Bootloader

Sozusagen der "zweite Teil" nach dem [Baseband](#) (daher auch "SPL" oder "Secondary Program Loader" genannt). Bleiben wir weiter bei den Hinke-Vergleichen, sind wir hier etwa im "Boot-Manager" (Lilo, Grub) gelandet (davor wäre noch der "MBR" oder "Master Boot Record" auf dem PC – das wäre hier der "IPL", der "Initial Program Loader" – der ist bei Androiden in Hardware gegossen, und daher nicht veränderbar).

Aber das wäre jetzt nur sehr grob und ungenau, denn hier steckt mehr drin: Der Android-Bootloader, sowie weitere Boot-Optionen wie das Recovery-Menü, Fastboot, u.a.m.

Brick

In der Regel das Lebensende eines Androiden – der dann nur noch als Briefbeschwerer o.ä. erhalten kann. Wörtlich heißt das zwar "Ziegelstein", aber das würde im Deutschen u. U. zu meilenweisen Verwechslungen mit gewissen Androiden aus dem Hause Motorola führen...

Was sich Google dabei dachte, als es die gleichnamige [Permission](#) einführte, sei der Fantasie anheim gestellt...

Wie verwandelt man einen Androiden nun in einen "Brick"? Dazu werde ich hier keine Schritt-für-Schritt-Anweisung geben (da wenig sinnvoll). Nur soviel sei gesagt: In etwa 95% aller Fälle hängt das mit dem [Flashen](#) eines zum Gerät inkompatiblen [RUU](#) zusammen. Vermeiden lässt sich solches also durch gründliches Lesen der Anleitungen und prüfen des "Zubehörs" **vor** dem "Brutzeln".

CalDav

CalDav steht für *Calendaring Extensions to WebDAV* und bezeichnet den nach [RFC 4791](#) spezifizierten Zusatz, der einen Zugriff auf Kalenderdaten über [WebDav](#) ermöglicht (siehe [Wikipedia](#)).

CSV

Diese Abkürzung steht für den englischen Begriff **Comma Separated Values**. Er beschreibt das Format einer reinen Textdatei zur Speicherung einfacher Tabellendaten (siehe auch [Wikipedia](#) für Details).

Custom ROM

Ein *Custom ROM* ist eine alternative Firmware, mit der man das vorinstallierte Android-System ersetzen kann – sofern man über [root](#)-Rechte auf dem Android-Gerät verfügt. Custom ROMs sind oftmals speziell auf die entsprechenden Geräte angepasst, und bieten in der Regel einen Mehrwert. Etwa eine aktuellere Android-Version, als der Hersteller sie anbietet. Oder weniger "Zwangsbeglückungen" (vorinstallierte Apps, die sich nicht ohne weiteres entfernen lassen). Auch zusätzliche Funktionalitäten sind hier keine Seltenheit.

Dalvik

Dafür muss ich ein klein wenig ausholen: Android besteht, vereinfacht gesagt, aus einem Linux-Kernel, auf dem eine spezielle Java-Version läuft. Letzteres ist die sogenannte *Dalvik VM* (wobei "VM" für "Virtual Machine" steht). Android Apps sind also in Java geschrieben.

Für die Ausführung der App wird der Java Code in einen sogenannten "Byte Code" übersetzt, der optimal auf die Hardware (und Android-Version) angepasst ist. Damit dies nicht bei jeder Ausführung der App geschehen muss, passiert diese "Übersetzung" unmittelbar nach der Installation der App – und der Byte-Code wird im sogenannten *Dalvik Cache* abgelegt. Da dies nach der Installation eines "neuen Systems" für alle Apps geschehen muss, dauert auch der erste Start nach der Neuinstallation ein wenig länger (dafür geht die Ausführung der Apps nachher entsprechend schneller).

Bei der Installation eines neuen [ROMs](#) muss aus genannten Gründen (optimale Anpassung ans System) der *Dalvik Cache* neu aufgebaut werden. Dafür gibt es im [Recovery-Menü](#) einen extra Menüpunkt – aber das ist im entsprechenden Kapitel auch erklärt.

Debuggen

Wörtlich "entkäfern". Ein Computer-antiker Begriff, der noch aus einer Zeit stammt, in der "Programmierung" durch das Ziehen von Drähten, Stecken von Röhren und Löten von Leiterbahnen stattfand. Da war der "Bug" im "Programm" nämlich durchaus wörtlich zu nehmen – wenn ein verbrutzelter Käfer für einen Kurzschluss sorgte...

Die Käfer sind mittlerweile zu groß geworden (oder vielmehr die Chips zu klein), trotzdem haben sich beide Begriffe gehalten: "Bug" für einen Fehler im Programm, und "Debuggen" für die Suche nach und das Entfernen desselben.

DES

[DES](#) steht für **Data Encryption Standard**. Es handelt sich dabei um einen symmetrischen Verschlüsselungsalgorithmus, der 1976 als offizieller Standard der US-Regierung bestätigt und seither international vielfach eingesetzt wird.

DHCP

Die Abkürzung steht für das **Dynamic Host Configuration Protocol**, welches in vielen Netzwerken zum Einsatz kommt. In WLANs ist es fast immer in Verwendung. Meldet sich ein Gerät beim DHCP-Server an, erhält es von diesem die für die Netzwerk-Kommunikation notwendigen Konfigurations-Parameter: Eine eigene IP-Adresse, die Adresse des zuständigen Routers, Nameserver, usw.. Detailliertere Informationen können u. a. [diesem Wikipedia-Artikel](#) entnommen werden.

DNS

Das Kürzel steht für das **Domain Name System**. Vereinfacht gesagt, handelt es sich dabei um eine Art "Telefonbuch" für das Internet, mittels welchem sich Domain- oder Rechner-Namen wie beispielsweise www.google.com in "Anschluss-Nummern" (die hier "IP-Adressen" heißen) wie in diesem Beispiel 209.85.148.104 übersetzen lassen. Wer sich für weitere Details interessiert, kann diese u. a. bei [Wikipedia](#) erfahren.

eMMC

Eine **e**mulierte **M**ulti**M**edia**C**ard: Gibt sich quasi als SD-Karte aus – tatsächlich handelt es sich jedoch um NAND-Flash.

Fastboot

Der Name ist zunächst ein wenig irreführend – handelt es sich hier doch nicht um die Möglichkeit, das Android-Gerät schneller einsatzbereit zu haben. *Fastboot* hat eigentlich mit dem installierten Android-Betriebssystem nicht einmal direkt etwas zu tun...

Zu finden ist ein Fastboot-Eintrag gelegentlich im [Boot-Menü](#). Und gedacht ist es in erster Linie zum schnellen Bearbeiten von [Partitionen](#) via USB. Dazu wählt man am Android-Gerät diesen Punkt aus, und kann dann vom PC aus mit der entsprechenden Software passende Befehle absetzen – etwa um die Daten auf einer Partition zu löschen, mit einer Image-Datei zu überschreiben, oder schlicht das Gerät neu zu starten.

Flashen

Den Androiden mit einem neuen [ROM](#) versehen – sei es ein "offizielles Firmware-Update", oder ein [Custom-ROM](#). Der Name rührt daher, dass hier die Daten größtenteils im "internen Speicher", dem sogenannten "Flash Speicher", landen.

FTP

Das **F**ile **T**ransfer **P**rotocol dient, wie der Name es sagt, der Übertragung von Dateien im Netz. Es gilt für diesen Einsatzbereich als besonders effizient und ressourcenschonend. Details lassen sich in der [Wikipedia](#) nachlesen.

GPG

GNU **P**rivacy **G**uard ([GPG](#)) ist ein auf [OpenPGP](#) basierendes Kryptographiesystem, welches u. a. [RSA](#)-Schlüssel empfiehlt und verwendet.

GPS

Das **G**lobal **P**ositioning **S**ystem wird genutzt, um per Kreuzpeilung verschiedener Satelliten den aktuellen Standort in bis zu vier Dimensionen zu bestimmen: Länge, Breite, Höhe und Zeit. Nähere Informationen finden sich in [diesem Wikipedia-Artikel](#).

IMEI

Die International Mobile Station Equipment Identity (IMEI) ist eine eindeutige 15-stellige Seriennummer, anhand derer jedes GSM- oder UMTS-Endgerät eindeutig identifiziert werden kann. ([Wikipedia](#)). Diese Nummer ist also Geräte-spezifisch, und wird von diversen Werbe-Modulen gern zur Identifizierung herangezogen. Mit ihr lässt sich aber auch ein Gerät beim Netzanbieter sperren, sodass ein Dieb es nicht mehr verwenden kann

(zumindest nicht im gesperrten Netz – dies weltweit durchzusetzen, dürfte ein wenig aufwendig sein)

Hardreset

Auch *Rücksetzen auf Werkseinstellungen* genannt: Wiederherstellung des Auslieferungs-Zustandes. Stimmt natürlich nicht so ganz, denn die ursprüngliche Firmware wird dabei nach einem Update nicht wieder hergestellt; es werden lediglich alle Nutzerdaten einschließlich vom Anwender installierter Apps etc. gelöscht.

HTTP

Hinter dieser Abkürzung verbirgt sich das **HyperText Transport Protocol**, welches zur (unverschlüsselten) Übertragung von Webseiten z. B. mit Web-Browsern verwendet wird. Erkennbar meist daran, dass die Adresse mit `http://` beginnt.

HTTPS

Klingt so ähnlich wie der vorige Punkt – und ist es auch: Das **HyperText Transport Protocol**, **Secure**, tut das gleiche – nur eben verschlüsselt, also für sicherere Übertragung. Erkennt man dann daran, dass die Adresse im Browser mit `https://` beginnt.

iCal

Gemeint ist hier das auf vKalendar basierende Datenformat [iCalendar](#). Dieses dient zum Austausch von Kalenderinhalten, und ist in [RFC 5545](#) standardisiert.

IntraNet

Ein privates, i. d. R. lokal begrenztes Netzwerk (siehe [Wikipedia](#)). Abgeleitet vom lateinischen "intra" ("innerhalb") und dem englischen "net" (Netzwerk), da es sich meist um ein firmeninternes Netz handelt.

Kernel

Da steckt das Wort "Kern" drin, genau. Wenn wir hier vom "Kernel" sprechen, meinen wir den "Betriebssystem-Kern", den "[Linux-Kernel](#)". Das ist, vereinfacht ausgedrückt, eine Abstraktions-Schicht: Unten speziell an die jeweilige Hardware angepasst, stellt der *Kernel* "oben" eine einheitliche Schnittstelle ([API](#)) für die Software zur Verfügung.

Bei Android läuft auf dem *Linux-Kernel* die [Dalvik-VM](#) (eigentlich je eine pro App), und in der *Dalvik-VM* sodann die [App](#).

KML

Die **Keyhole Markup Language** ist ein spezieller [XML](#)-Dialekt, der zum Austausch geografischer Informationen verwendet wird. Die bekanntesten Vertreter, die dieses Format verwenden, sind *Google Maps* und *Google Earth*. Eine Spezialform davon ist *KMZ*: Hier steht das "Z" für "ZIP", die Datei ist also komprimiert. Auf diese Weise ist es möglich, zusätzliche Elemente wie Icons mit der KML-Datei beisammen zu halten.

MAC-Adresse

Die *MAC-Adresse (Media-Access-Control-Adresse)* ist die Hardware-Adresse jedes einzelnen Netzwerkadapters, die zur eindeutigen Identifizierung des Geräts in einem Rechnernetz dient. Bei Apple wird sie auch *Ethernet-ID*, *Airport-ID* oder *Wi-Fi-Adresse* genannt, bei Microsoft *Physikalische Adresse*. ([Wikipedia](#))

NAND-Flash

Daraus besteht der interne Speicher unserer Androiden: Flash-Speicher, der in der sogenannten NAND-Technik gefertigt ist. Genauere Details dazu lassen sich u. a. der [Wikipedia](#) entnehmen.

Nandroid Backup

Ein vollständiges System-Backup, welches sich z. B. aus dem [Recovery-Menü](#) heraus erstellen und auch wieder herstellen lässt. Hier werden nicht einzelne Dateien gesichert, sondern ein Abbild ("Image") des gesamten Systems wird angelegt. Es ist also ein "Alles-oder-Nichts": Die Wiederherstellung einzelner Dateien ist hier nicht vorgesehen (auch wenn dies mit [Titanium Backup](#) mittlerweile möglich ist).

Insbesondere bevor man ein [Custom-ROM](#) einspielt, aber auch generell vor einem System-Update sollte ein Nandroid-Backup angelegt werden. Es ist natürlich auch sonst immer eine gute Idee, ein komplettes Backup zur Hand zu haben.

NAS

Network Attached Storage – oder, stark vereinfacht ausgedrückt: Festplatte mit Netzwerk-Anschluss "for the Housegebrauch". Mittlerweile gibt es NAS-Systeme mit einer Kapazität von 2 TB bereits für weniger als 200 Euro.

NFS

Wie der Name es vermuten lässt, erlaubt das **Network File System** den Dateizugriff über das Netzwerk – ähnlich wie bei einer Windows-Laufwerks-Freigabe. Dieses Protokoll kommt hauptsächlich unter Linux/Unix zum Einsatz. Nähere Informationen können u. a. der [Wikipedia](#) entnommen werden.

Notification Area

Die *Notification Area* ist auch als "Statusbereich" bekannt, und wird durch die Statusleiste am oberen Bildschirmrand repräsentiert. Zieht man diese Leiste nach unten, öffnet sich die "Area" mit detaillierteren Informationen, welche Apps dort hinterlegen können.

OOM-Killer

Der Name lässt etwas "Böses" vermuten – doch der Benannte sorgt für Stabilität: Es handelt sich um den System-Prozess, der – sobald der Arbeitsspeicher ([RAM](#)) knapp wird (OOM: **O**ut **O**f **M**emory) – wieder für Platz sorgt. Indem er "Altlasten" entsorgt: Apps, die nur im Hintergrund rumliegen und länger nicht mehr benutzt, oder gar vom Benutzer bereits beendet wurden, beispielsweise. Mit [root](#)-Rechten ausgestattet, kann man auch konfigurieren, wie viel Platz er mindestens freihalten soll.

OTA

Da liegt was in der Luft... Denn OTA steht für "**O**ver **T**he **A**ir". Ja was denn? Beim Rundfunk ist es "On The Air" und heißt Musik. Bei Phil Collins "In The Air Tonight". Und bei Android ein "(komplettes) Over The Air Update" – also ein [Kotau](#), sozusagen. Die Frage wäre da nur, wer dabei der Kaiser ist...

Also, kurz gefasst: Beim OTA werden Software-Updates des Herstellers/Providers über das Funknetz des letzteren verteilt.

Partition

Eine Partition ist ein zusammenhängender Bereich auf einem Datenträger (unter Windows häufig mit einem Laufwerksbuchstaben verbunden).

Auf einem Android-System sind immer mehrere Partitionen in Benutzung, auch wenn nur der interne Speicher zur Verfügung steht (und keine SD-Karte eingelegt ist): So ist das "/system" in der Regel nur lesend eingebunden (um Änderungen im Betrieb zu verhindern), während für [Apps](#) und Daten eine eigene Partition ("/data") bereitsteht.

Die SD-Karte beinhaltet meist nur eine (in der Regel unter "/sdcard" eingebundene) Partition. Es sind aber auch hier mehrere Partitionen möglich (und werden z. B. mit [App2SD+/Link2SD](#) auch verwendet) – wobei Android bei Anschluss an den PC via USB nur jeweils die erste Partition davon freigibt.

Weitere Details können z. B. bei [Wikipedia](#) nachgelesen werden.

Peer-to-Peer

Das englische Wort "peer" bedeutet soviel wie "Gleichgestellter". Teilnehmer in einem Peer-to-Peer-Netzwerk sind sich also "gleichgestellt" – im Gegensatz zu einem "Client-Server-Modell". Jeder Teilnehmer kann also sowohl Dienste eines anderen in Anspruch nehmen, als auch eigene Dienste bereitstellen.

Mehr dazu kann auch in [diesem Wikipedia-Artikel](#) nachgelesen werden.

Permission

...meint hier die im Android-System definierten Zugriffsberechtigungen, wie im Kapitel [Zugriffsrechte](#) erklärt. Eine Übersicht dazu findet sich im Anhang [Google Permissions](#).

Proxy

Ein Vermittler (von lateinisch *proximus*, der Nächste). Gemeint ist hier ein Service, der ihm übergebene Anfragen weiterleitet – da sie beispielsweise auf direktem Wege nicht zustellbar sind. Denkbar ist auch, dass ein Proxy Daten für den schnelleren Zugriff optimiert – meist indem er selbige zwischenspeichert ("caching proxy"). Er könnte aber auch, wie im Falle des Web-Browsers *Opera* oder des "Mobilizers" von *Google* Bilder für mobile Geräte herunterrechnen, um so den Traffic zu minimieren (und die Übertragung zu beschleunigen). Weitere Details finden sich u. a. in der [Wikipedia](#).

RAM

Diese drei Buchstaben stehen für **R**andom **A**ccess **M**emory – also Speicher, auf den man nach Belieben an beliebiger Stelle zugreifen kann. Im Gegensatz nicht etwa zu [ROM](#), sondern zu Dingen wie Bandlaufwerken (jaja, so alt ist der Begriff schon), bei denen man sich erst mühsam vom Start zur gewünschten Position (linear) vortasten muss.

Sowohl auf PCs wie auch auf unseren Androiden ist damit meist der Arbeitsspeicher gemeint, in den die Programme/Apps zur Ausführung geladen werden. Üblicherweise ist dies der Bereich, der generell zu klein ist... oder von dem man halt nie genug haben kann...

Recovery Menü

Ein separater Bereich des Boot-Menüs, aus dem heraus verschiedene Operationen wie [Nandroid-Backup](#) oder auch das Bereinigen des [Dalvik-Caches](#) möglich sind.

In das *Recovery Menü* gelangt man in der Regel durch eine spezielle Tasten-Kombination beim Einschalten. Diese ist aber zumindest von Hersteller zu Hersteller unterschiedlich. Bei HTC ist es üblicherweise das Halten der "Leiser-Taste" während des Einschaltens. Bei Motorolas Milestone muss die Kamera-Taste beim Einschalten gedrückt gehalten, und anschließend die "Lauter-Taste" betätigt werden. Und so weiter. Bei Bedarf also am besten im Forum erlesen/erfragen.

Reguläre Ausdrücke

Ein *regulärer Ausdruck* lässt sich am besten als Zeichenkette mit Suchmustern beschreiben. Dank komplexer Regeln lässt sich damit recht gezielt auch in größeren Textmengen suchen. Genauere Details lassen sich beispielsweise [diesem Wikipedia-Artikel](#) entnehmen.

ROM

Richtig offensichtlicher Mist ist diese **real** offerierte **Mehrdeutigkeit**: Manchmal hat man den Eindruck, er wurde nur zur Verwirrung der Massen eingeführt. Wer einmal den Namen für eine Android-Komponente nicht kennt, sagt einfach "ROM". Klingt, als wüsste man voll Bescheid – und die Chance, dass das auch noch Sinn ergibt, ist verdammt hoch...

Aber im Ernst: Worum geht es hier? Eigentlich steht der Begriff "ROM" für **Read Only Memory** – also Speicher, auf den ausschließlich lesend zugegriffen werden kann. Ja, richtig: So wie bei CD-ROM, da steckt das ja auch drin. Nur bei Android, da kann das alles mögliche sein. Nicht selten sachlich falsch – aber wen kümmert's? Schauen wir uns also die einzelnen Bedeutungen einmal an:

Systemspeicher: Teile des Android-Systems werden in der Tat "nur lesend" eingebunden. Unter anderem eine Schutzmaßnahme, um Veränderungen zu erschweren (damit wir die dusseligen Apps, mit denen uns die Hersteller/Provider "beglücken", nicht einfach löschen können). So heißt es z. B. in den Spezifikationen des HTC Wildfire: "384 MB RAM; 512 MB ROM". Nonsens(e): Ein Blick hinter die Kulissen offenbart, dass nur 250MB read-only (/system) eingebunden sind. Die restlichen 250MB "ROM" stehen zur Installation von Anwendungen zur Verfügung. Read-only? Mitnichten. De facto kann der ganze Bereich jederzeit schreibbar gemacht werden, sofern man [root](#) hat. Also eher irreführend – richtiger müsste es hier heißen: "interner Speicher", oder – in Abgrenzung vom [RAM](#) – "interner Flash-Speicher".

Das System selbst: Um die Verwirrung komplett zu machen, wird auch hier gern von "ROMs" gesprochen. Das hat schon Tradition: Auch bei älteren Spiele-Konsolen sprach man davon, "ein ROM zu laden". Hier war es aber "seinerzeit" tatsächlich eine Cartridge – also ein Speicher-Chip, den man an das Gerät ansteckte. Später kamen dann die Emulatoren, welche die "antiken Geräte" auf moderner Hardware emulieren können. Und hier kommt diese "Cartridge" natürlich in Form einer Datei daher. Den Begriff "ein ROM laden" hat man beibehalten. Und schließlich weiter übertragen...

root

Aus Herstellersicht: Die Wurzel allen Übels. Objektiv betrachtet: Der Administrator (auch "SuperUser") eines Linux-Systems. Der darf alles, und kann alles (kaputtmachen auch, ja).

RSA

***RSA** ist ein asymmetrisches kryptographisches Verfahren, das sowohl zur Verschlüsselung als auch zur digitalen Signatur verwendet werden kann (Wikipedia). Hierbei kommt ein Schlüsselpaar (privater und öffentlicher Schlüssel) zum Einsatz. Der Name des Verfahrens setzt sich aus den Anfangsbuchstaben der Familiennamen seiner Erfinder zusammen: **R**ivest, **S**hamir, **A**dleman.*

RUU

Radio Unit Update: Eine Art [update.zip](#) für das [Radio-Image](#), also ein "Firmware-Upgrade".

ROM Upgrade Utility: Wie der Name bereits sagt, ein Utility zum Upgrade des [ROM](#), welches entweder vom Hersteller oder von Drittanbietern zur Verfügung gestellt und vom PC aus installiert wird.

Wenn nicht ganz klar ist, was von beidem gemeint ist, ist es in der Regel das erste – wobei das durchaus mit dem zweiten identisch sein kann, da die Begriffe oftmals gleichbedeutend verwendet werden. Was mancherorts als "ROM Upgrade Utility" bezeichnet wird, ist nämlich nichts anderes als das Update des Radio-Images...

Salt

Salt (deutsch Salz) bezeichnet in der Kryptographie eine zufällig gewählte Zeichenfolge, die an einen gegebenen Klartext vor der Verwendung als Eingabe einer Hashfunktion angehängt wird, um die Entropie der Eingabe zu erhöhen. (Wikipedia)

SDK

Das SDK ist die Grundlage für die Android App Entwicklung und liegt für die jeweilige Version von Android vor. Es ist aber nicht nur dafür verwendbar, sondern bringt auch einige brauchbare Tools wie [Fastboot](#), den [Dalvik](#) Debug Monitor sowie [ADB](#) mit.

Ergänzt man das Ganze noch um [Eclipse](#), kann es mit der Entwicklung von [Apps](#) losgehen!

SIP

Das **S**ession **I**nitiation **P**rotokoll findet in der Internet-Telefonie häufige Anwendung, und wird im allgemeinen Sprachgebrauch teilweise gar mit "VoIP" (Voice-over-IP, also der Internet-Telefonie selbst) gleichgesetzt. Nähere Informationen finden sich u. a. bei der [Wikipedia](#).

SMB

Server **M**essage **B**lock (kurz **SMB**, teils auch als LAN-Manager- oder NetBIOS-Protokoll bekannt) ist ein Kommunikationsprotokoll für Datei-, Druck- und andere Serverdienste in Netzwerken. Es ist der Kern der Netzwerkdienste von Microsofts LAN Manager, der Windows-Produktfamilie sowie des LAN Servers von IBM. Weiter wird es von den frei verfügbaren Softwareprojekten Samba und Samba-TNG verwendet, um Windows-Systemen den Zugriff auf

Ressourcen von UNIX-basierten Systemen zu ermöglichen und umgekehrt. ([Wikipedia](#)) Unter Linux/Unix wird das Kürzel übrigens mit zwei Vokalen angereichert, damit man es leichter aussprechen kann: SaMBa.

Sniffer

Ein Sniffer (engl. to sniff, riechen, schnüffeln) ist eine Software, die den Datenverkehr eines Netzwerks empfangen, aufzeichnen, darstellen und ggf. auswerten kann. Es handelt sich also um ein Werkzeug der Netzwerkanalyse. ([Wikipedia](#))

SSH

Das Kürzel *SSH* bezeichnet die **S**ecure **S**hell – was zum Einen ein Netzwerk-Protokoll, zum Anderen aber auch entsprechende Programme für die Herstellung/Nutzung einer sicheren Datenverbindung benennt. Es kommt i. d. R. eine starke Verschlüsselung zum Einsatz, die auch bereits beim Verbindungsaufbau und der Anmeldung (Login) gilt. Eine detailliertere Beschreibung kann der [Wikipedia](#) entnommen werden.

SSI

Server-**S**ide **I**ncludes bezeichnet einfache, i. d. R. in HTML-Dokumente eingebettete Skript-Befehle, welche der Webserver vor der Auslieferung der Seite an den Client interpretiert und durch entsprechende Inhalte ersetzt. Für weiterführendes, siehe [Wikipedia](#).

SSID

Diese Abkürzung steht für den Begriff **S**ervice **S**et **I**dentifier, und sie bezeichnet den frei wählbaren Namen eines WLAN-Netzes. Für Details, siehe [Wikipedia](#).

Symlink

Eine symbolische Verknüpfung, auch symbolischer Link, Symlink oder Softlink genannt, ist eine Verknüpfung in einem Dateisystem (Dateien und Verzeichnisse), die auf eine andere Datei oder ein anderes Verzeichnis verweist. Es ist also lediglich eine Referenz auf die Zieldatei bzw. das Zielverzeichnis. ([Wikipedia](#))

Tethering

Das Bereitstellen einer bestehenden Netzverbindung für andere Geräte. In unserem Kontext handelt es sich dabei in der Regel um eine mobile Datenverbindung, die per WLAN, Bluetooth, oder USB weitergegeben wird.

TAR

Kurzform für **T**ape **A**rchive – denn dafür wurde dieses Format ursprünglich eingesetzt. Obwohl Bandlaufwerke heutzutage weitgehend zum "alten Eisen" gehören, ist das Format (zumindest in der Unix- und Linux-Welt) nach wie vor sehr beliebt – und kommt meist in Verbindung mit einem Kompressionsprogramm wie GZip oder BZip daher. Nähere Details lassen sich u. a. [diesem Wikipedia-Artikel](#) entnehmen.

TOR

Das hat jetzt nichts mit Fußball zu tun, und ich bin auch nicht Balla-Balla. Wer genau hinschaut erkennt, dass alle drei Buchstaben groß geschrieben sind – ganz offensichtlich handelt es sich hier also um eine Abkürzung.

Und die steht für **The Onion Router**. Zwiebel-Ruten? Gedacht ist hier an die unzähligen Schalen oder Schichten, die eine Zwiebel hat (und vielleicht auch an die Tränen, die ein Auseinandernehmen derselben verursacht). Beim **TOR-Netzwerk** steht jede Schicht hier für eine Erhöhung der Anonymität: Je mehr Router beteiligt sind, desto schwieriger lässt sich der Absender eines Paketes ausmachen.

Für eine genauere Beschreibung möchte ich jedoch auch hier wieder auf den zugehörigen [Wikipedia-Artikel](#) verweisen.

Traces

Die Ablaufverfolgung (englisch tracing) bezeichnet in der Programmierung eine Funktion zur Analyse von oder Fehlersuche in Programmen. [...] Dabei wird z. B. bei jedem Einsprung in eine Funktion, sowie bei jedem Verlassen eine Meldung ausgegeben, sodass der Programmierer mitverfolgen kann, wann und von wo welche Funktion aufgerufen wird. (Wikipedia)

Tracker

Die Rede ist hier nicht vom "Trecker", der laut knatternd die Landstraßen verstopft und eigentlich aufs Feld gehört. *Tracker* leitet sich vielmehr vom englischen Wort "track", also "Spur" ab. Ein *Tracker* in unserem Kontext versucht, Nutzerspuren aufzuzeichnen. Klassische Beispiele sind die [Web-Bugs](#). Nicht verwechseln: Im Bereich "GPS & Navigation" gibt es auch "Tracker", die Spuren aufzeichnen – hier ist das aber definitiv gewünscht, um anschließend z. B. die zurückgelegte Route auf einer Karte ansehen zu können.

Tunnel

Ein *Tunnel* bündelt die Netzwerk-Kommunikation eines oder mehrerer Protokolle, und bettet diese ggf. in ein anderes Protokoll ein. Am Tunnelein- und Ausgang wird dabei das "originale Protokoll" (z. B. HTTP zum Abruf einer Webseite) benutzt, während die Übertragung im Tunnel Verschlüsselt (beispielsweise über SSH) abläuft. Details dazu finden sich u. a. in der [Wikipedia](#).

Update.Zip

Ganz offensichtlich eine Datei. Und ebenso offensichtlich will diese etwas aktualisieren – nur was?

Es handelt sich hier um ein "[flashbares](#) Update". Offizielle Firmware-Updates kommen meist unter diesem Namen daher. Und da das System eine solche Datei, so sie im Wurzel-Verzeichnis der SD-Karte liegt, als ein solches betrachtet, lässt sich auf diese Weise auch so einiges anderes ins System mogeln. Das nutzt z. B. [Titanium Backup](#) aus, wenn es ein update.zip erstellt.

Einspielen lässt es sich zum Beispiel über das [Bootmenü](#).

VNC

Virtual Network Computing, kurz **VNC**, ist eine Software, die den Bildschirminhalt eines entfernten Rechners (Server) auf einem lokalen Rechner (Client) anzeigt und im Gegenzug Tastatur- und Mausbewegungen des lokalen Rechners an den entfernten Rechner sendet. Damit kann man auf einem entfernten Rechner arbeiten, als säße man direkt davor. ([Wikipedia](#))

VPN

Ein **V**irtuelles **P**rivates **N**etzwerk bindet einen Computer (oder, in unserem Fall, einen Androiden) über eine verschlüsselte Verbindung in ein entferntes Netzwerk (Firmennetz, Uni-Netz, heimisches Netzwerk) ein, als wäre er "dort vor Ort" (näheres dazu bei [Wikipedia](#)).

Web-Bug

Kleine Wanzen, die sich auf diversen Webseiten verstecken, um Besucher-Informationen zu sammeln. Mancherorts werden sie auch verharmlosend "Zählpixel" genannt.

Man mag sich fragen: Welchen Schaden kann so ein 1x1 Pixel großes Bildchen denn anrichten? Kommt ganz darauf an. Denn beim Abruf desselbigen vom Server wird ja nicht nur diese kleine Grafik-Datei zum Benutzer übertragen. Auch in der Gegenrichtung fließen so einige Informationen: Auf welcher Webseite war das Bild eingebunden (Referrer)? Welche IP-Adresse/Browser/Betriebssystem/... hat der Anwender im Einsatz? Wo war er noch (Kekse/Cookies helfen gern dabei, Lücken zu schließen).

Da kann so einiges zusammen kommen, und schon fast ein Profil des Anwenders erstellt werden. Mehr Details gefällig? Wie immer, hilft [Wikipedia](#) hier gern weiter...

WebDAV

WebDAV steht für *Web-based Distributed Authoring and Versioning* und ist ein offener Standard zur Bereitstellung von Dateien im Internet, über den auf Daten wie bei einer Online-Festplatte zugegriffen werden kann. Dabei ist auch eine Versionskontrolle spezifiziert. In der Regel wird *WebDAV* über einen Webserver umgesetzt, da es sich hier um eine Erweiterung des HTTP-Protokolls handelt. Nähere Details finden sich u. a. in der [Wikipedia](#).

Wipe

Wörtlich übersetzt: Löschen. Das "was" ist hier allerdings die Frage. Und da kommt es darauf an, wen man fragt bzw. wie man den Wipe initialisiert.

Hardreset: Eine Form des Wipe ist mit diesem gleichbedeutend, denn sie löscht lediglich die `/data` [Partition](#). Das ist der Bereich, in dem die selbst installierten Apps sowie die Daten abgelegt werden.

Dalvik-Cache: Mit dem Wipe/Löschen des Dalvik-Caches erzwingt man eine Neu-Übersetzung des Programmcodes aller Apps. Damit geht kein Datenverlust einher: Lediglich der nächste Gerätestart dauert etwas länger.

Komplett-Wipe: Den gibt es in verschiedenen Versionen des [Recovery-Menüs](#). Sollte nur ausgeführt werden, wenn man auch wirklich weiß, was man da tut: Der löscht nämlich alle Daten von allen Partitionen, auch von `/system`. Danach geht dann nichts mehr – es lässt sich lediglich ein neues [ROM](#) einspielen.

XML

Die *eXtensible Markup Language* stellt hierarchisch strukturierte Daten in Form von Textdaten dar, und wird u. a. für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt. Normalerweise enthalten XML-Dateien keinen Binärcode, auch wenn dies aufgrund der Erweiterbarkeit durchaus möglich ist.

Daher kann man selbige notfalls auch im "Texteditor" betrachten – wenn auch nicht unbedingt sehr komfortabel.
Weiterführende Informationen finden sich bei der [Wikipedia](#).

Google Permissions - und was sie bedeuten

Normalerweise sieht man eine Kurzbeschreibung der Permission (z. B. bei der Installation einer App). Die technische Bezeichnung taucht selten im Klartext für den Anwender auf – man kann aber z. B. auch einen Blick auf das [Manifest](#) werfen, und da stehen sie im Klartext.

Nun werde ich aber hier nicht alle Permissions auflisten, das wäre einfach zu viel. Die findet man bei Interesse im [Entwickler-Handbuch](#) (allerdings auf Englisch). Ein paar ausgewählte, die doch häufiger einmal auftauchen könnten, möchte ich aber hier kurz erklären:

Permission	Erklärung
A CESS_COARSE_LOCATION	<i>Ungefährer (netzwerkbasierter) Standort.</i> Hier kommt kein GPS zum Einsatz, sondern die Informationen von Funkmasten (Cell-ID) sowie WLANs
ACCESS_FINE_LOCATION	<i>Genauer (GPS-) Standort.</i> Exakte, per GPS ermittelte Standortdaten.
ACCESS MOCK_LOCATION	<i>Falsche Standortquellen für Testzwecke.</i> Fake Location (falsche Standortdaten vom System anfordern). Für Testzwecke gedacht (z. B. im Emulator); wird aber scheinbar auch benötigt, um ein externes GPS Gerät nutzen zu können. Hier geht es nicht darum, einer anderen App eine Fake-Location unterzujubeln – das ginge allenfalls in Verbindung mit <code>INSTALL_LOCATION_PROVIDER</code> .
ACCESS_NETWORK_STATE	<i>Netzwerkstatus anzeigen.</i> Informationen über Netzwerke (besteht eine Verbindung, und wenn ja zu welchem Netzwerk?)
ACCESS_SURFACE_FLINGER	Zugriff auf die API des "Surface Flinger" (Teil des Medien-Frameworks unter Android: Stellt einen systemweiten "Oberflächen-Kompositor" bereit, der sich um das Rendering in Framebuffer-Devices kümmert - also Grafik, Grafik-Beschleunigung und so).
ACCESS_WIFI_STATE	<i>WLAN-Status anzeigen.</i> Informationen über WiFi-Netzwerke (besteht eine Verbindung, und wenn ja zu welchem Netzwerk? Welche Netzwerke sind verfügbar?)
ACCOUNT_MANAGER	<i>Als Konto-Manager fungieren.</i> App darf mit Konto-Authentifizierern interagieren. (für Systemanwendungen reserviert).
AUTHENTICATE_ACCOUNTS	<i>Als Kontoauthentifizierer fungieren.</i> Konto-Authentifizierungsfunktionen verwenden, Konten erstellen, Abrufen und Einstellen der zugehörigen Passwörter.
B IND_APPWIDGET	<i>Widgets auswählen.</i> Erlaubt einer App dem AppWidget-Service mitzuteilen, welche App auf die AppWidget-Daten zugreifen darf. Mit dieser

Permission	Erklärung
	Berechtigung kann anderen Anwendungen Zugriff auf persönliche Daten gewährt werden. Laut API-Referenz sollten nur sehr wenige Apps diese Permission benötigen.
BLUETOOTH	<i>Bluetooth-Verbindungen herstellen.</i> Zugriff auf bereits "authorisierte" Bluetooth-Geräte
BLUETOOTH_ADMIN	<i>Bluetooth-Verwaltung.</i> Bluetooth-Geräte "autorisieren" (also "pairen" und so). Eine mit dieser Permission ausgestattete App darf selbständig Bluetooth-Verbindungen aufbauen und etablieren – auch zu "wildfremden" Geräten.
CALL_PHONE	<i>Telefonnummern direkt anrufen.</i> Anruf ohne Bestätigung durch den Anwender tätigen. Wird z. B. für Kontakt-Widgets benötigt, wenn ein "Tapp" auf selbige direkt einen Anruf auslösen soll – macht aber bei einer Mal-App herzlich wenig Sinn. Gilt nicht für Notruf-Nummern.
CALL_PRIVILEGED	<i>Alle Telefonnummern direkt anrufen.</i> Wie CALL_PHONE, aber inklusive Notruf-Nummern ("Hallo, Polizei – Anwender ist gerade in Bank eingebrochen... Bitte Fußboden reparieren...")
CAMERA	<i>Fotos aufnehmen.</i> Vollzugriff auf die Kamera. Nebenwirkung: Diese App lässt sich nicht auf Geräten installieren, die über keine Kamera verfügen. Die API-Referenz schreibt sinngemäß: "Wenn die App auch ohne Kamera bedienbar ist, diese Permission nicht anfordern." Da sei die Frage erlaubt: Braucht man sie dann überhaupt, wenn es auch ohne geht?
CHANGE_CONFIGURATION	<i>UI-Einstellungen ändern.</i> Änderungen an der Umgebung durchführen. API-Ref nennt als Beispiel "Locale", also Ländereinstellungen wie Währung und Zeitformat. Beschreibung sehr vage.
CHANGE_NETWORK_STATE	<i>Netzwerkonnktivität ändern.</i> Netzwerk-Status ändern (also z. B. Verbindung trennen)
CHANGE_WIFI_MULTICAST_STATE	<i>WLAN-Multicast-Empfang zulassen.</i> Damit können Datenpakete an mehrere Empfänger zeitgleich verschickt werden, ohne dass dies zusätzliche Bandbreite erfordert. Macht z. B. Sinn bei einem Streaming-Server, der mehrere Clients bedient ("Radio"). Gleichzeitig ermöglicht dies auch den Empfang von Netzwerk-Paketen, die nicht an das eigene Gerät gerichtet sind (Netzwerk-Sniffer).
CHANGE_WIFI_STATE	<i>WLAN-Status ändern.</i> CHANGE_NETWORK_STATE für WiFi. Kann auch

Permission	Erklärung
	Änderungen an konfigurierten WLAN-Netzen vornehmen.
CLEAR_APP_CACHE	Alle Cache-Daten der Anwendung löschen. Cache beliebiger/aller Anwendungen leeren
CLEAR_APP_USER_DATA	Alle Cache-Daten der Anwendung löschen. Benutzerdaten beliebiger/aller Apps löschen (siehe Einstellungen→Anwendungen verwalten, der Button "Daten löschen" bei jeder Anwendung) – richtiger wäre also Anwendungsdaten löschen oder CLEAR_APP_DATA, das "User" verwirrt hier ein wenig.
DELETE_CACHE_FILES	(einzelne) Dateien aus dem Cache löschen
DELETE_PACKAGES	Anwendungen löschen. Apps entfernen/ löschen/deinstallieren/wegmachen
DEVICE_POWER	Tiefgreifender (low-level) Eingriff in die Energieverwaltung (Power Management). Hat nicht direkt etwas mit "Power Off" zu tun, könnte aber bei Missbrauch durchaus zu selbigem führen...
DIAGNOSTIC	Lese-/Schreibberechtigung für zu Diagnosegruppe gehörige Elemente. Lese- und Schreibzugriff auf "diagnostic resources" – die API-Referenz beschreibt leider nichts genaueres.
DISABLE_KEYGUARD	Tastensperre deaktivieren. Tastensperre (inkl. deren Passwort-Schutz) deaktivieren, sodass der Bildschirm nicht mehr automatisch gesperrt wird. Sinnvoll z. B. bei Video-Apps und insbesondere bei Navis – und bei eingehenden Telefonaten.
EXPAND_STATUS_BAR	Statusleiste ein-/ausblenden. Status-Bar (Notification?) erweitern/kollabieren. Wohl die Lite-Version von STATUS_BAR
GET_ACCOUNTS	Bekannte Konten suchen. Liste konfigurierter Accounts abrufen (nur die Accounts, nicht die Zugangsdaten selber). Mit dieser Permission kann lediglich festgestellt werden, welche Accounts existieren.
GET_TASKS	Laufende Anwendungen abrufen. Informationen über laufende Anwendungen abrufen. Wird natürlich von Task-Managern und -Killern, aber auch von Akku-Statistik-Apps benötigt. "Böse Apps" können dies nutzen um auszukundschaften, wo sich lohnende Daten zum Klauen finden lassen.
INJECT_EVENTS	Tasten und Steuerungstasten drücken. "Generieren" und "ausführen" bestimmter Events, wie z. B. Benutzer-Eingaben. Die App

Permission	Erklärung
	kann also vermutlich andere Apps "fernbedienen".
INSTALL_LOCATION_PROVIDER	App will selber Ortsdaten bereitstellen ("Du bist jetzt <i>hier</i> "). Woher sie die nehmen will? Naja, vielleicht von einem Bluetooth-GPS o.ä.
INSTALL_PACKAGES	<i>Anwendungen direkt installieren.</i> Andere Apps installieren. Kann OK sein (App-Manager), muss aber nicht (Wallpaper etc. wollen vielleicht eher Schadsoft nachladen, wenn sie diese Permission anfordern)
INTERNET	<i>Uneingeschränkter Internetzugriff.</i> Öffnen von Netzwerk-Sockets. Die App kann also beliebige Internet-Verbindungen herstellen. Wird von allen Apps gebraucht, die Werbung anzeigen wollen – aber auch von Web-Browsern, Mail-Apps, u. a. m..
KILL_BACKGROUND_PROCESSES	<i>alle Anwendungen im Hintergrund schließen.</i> Hintergrund-Prozesse "töten", also beenden. Dabei kann es sich um die eigenen Prozesse handeln (was dem Anwender die Möglichkeit gibt, das Programm tatsächlich zu beenden, statt es nur in den Hintergrund zu schieben) – es können aber eben so gut fremde Prozesse beendet werden. I.d.R. handelt es sich dann um einen Task-Manager oder Task-Killer .
MANAGE_ACCOUNTS	<i>Als Konto-Manager fungieren.</i> Accounts/ Zugangsdaten <i>verwalten</i> – also auch verändern. Die Doku ist leider wieder einmal sehr vage – aber hier würden bei mir die Alarmglocken läuten.
MODIFY_PHONE_STATE	<i>Telefonstatus ändern.</i> Status der Telefonie anpassen: Power, MMI-Codes (z. B. Rufumleitung [de]aktivieren, Rufnummernübermittlung ein/ausschalten) etc. – jedoch nicht Anrufe tätigen. Allerdings kann das Netzwerk (zu einem anderen Anbieter, Roaming) gewechselt oder die Mobilfunkverbindung ein- bzw. ausgeschaltet werden, ohne dass der Benutzer davon informiert wird.
MOUNT_FORMAT_FILESYSTEMS	<i>Externen Speicher formatieren.</i> Externe Dateisysteme (SD-Karten etc.) <i>formatieren</i> (Vorsicht! Nach dem Formatieren ist das entsprechende Dateisystem leer, die (vorher) darauf befindlichen Daten sind weg!). Nix für Wallpaper, Spiele, etc.!
MOUNT_UNMOUNT_FILESYSTEMS	<i>Dateisysteme bereitstellen oder Bereitstellung aufheben.</i> Dateisysteme ein- und ausbinden.

Permission	Erklärung
	Toll für externe Festplatten am Telefon – gilt aber ebenso für SD-Karten.
PROCESS_OUTGOING_CALLS	<i>Abgehende Anrufe abfangen.</i> Ausgehende Anrufe beobachten, verändern oder abbrechen. Hm, könnte das einen Anruf bei der Mailbox ins Ausland weiterleiten? A propos Mailbox: Eine entsprechende Permission für eingehende Anrufe konnte ich bislang nicht entdecken.
READ_CALENDAR	<i>Kalenderdaten lesen.</i> Sollte klar sein: Alle Termine können damit gelesen werden.
READ_CONTACTS	<i>Kontaktdaten lesen.</i> Damit ist das Adressbuch fällig.
READ_FRAME_BUFFER	Zugriff auf die Frame-Buffer Daten (vereinfacht gesagt: Den Inhalt des Bildschirms). Erlaubt u. a. das Erstellen von Screenshots. Da stellt sich die Frage: Warum gibt es dann keine App zur Erstellung von Screenshots, die keine root-Rechte benötigt?
READ_HISTORY_BOOKMARKS	Erlaubt lesenden Zugriff auf Lesezeichen und Browserverlauf (Chronik).
READ_LOGS	<i>System-Protokolldateien lesen.</i> Lesender Zugriff auf die Log-Dateien des Systems. Hier werden allgemeine Informationen zu durchgeführten Aktionen gespeichert, i. d. R. jedoch keine vertraulichen Informationen (es sei denn, ein Programmierer hat etwas verbockt).
READ_OWNER_DATA	<i>Eigentümerdaten lesen.</i> Auslesen der auf dem Gerät gespeicherten Eigentümerdaten.
READ_PHONE_STATE	<i>Telefonstatus lesen und identifizieren.</i> Zugriff auf die Telefonfunktionen des Gerätes. Eine Anwendung mit dieser Berechtigung kann unter anderem die Telefon- und Seriennummer dieses Telefons ermitteln und feststellen, ob ein Anruf aktiv ist oder mit welcher Nummer der Anrufer verbunden ist. Ein Media-Player kann so z. B. bei eingehenden Anrufen automatisch auf "Pause" schalten (dafür benötigt er diese Permission aber nur unter Android 1.6 oder früher – ab Version 2.0 gilt diese Ausrede nicht mehr). Bei Werbung (z. B. AdMob) wird dies häufig zum Auslesen der IMEI/IMSI genutzt um festzustellen, welche Werbung auf dem Gerät bereits angezeigt wurde (eindeutige Identifizierung, Tracking). Apps, die auch für Android 1.6 und früher kompatibel sein sollen, wird diese Permission automatisch gesetzt.
READ_SECURE_SETTINGS	Lesezugriff auf Systemeinstellungen (u. a. Umschalter für die mobile Datenverbindung). Eigentlich dem System vorbehalten.

Permission	Erklärung
READ_SMS	<i>SMS oder MMS lesen.</i> Damit lassen sich bereits gespeicherte Kurznachrichten lesen. Darunter können natürlich auch vertrauliche Informationen sein...
READ_SYNC_SETTINGS	<i>Synchronisierungseinstellungen lesen.</i> Lesezugriff auf die Einstellungen der Synchronisation – etwa um festzustellen, ob selbige für Kontakte aktiviert ist. Ein gutes Zeichen: Die App möchte vielleicht wissen, ob der Anwender eine Datensynchronisation im Hintergrund erlaubt, und sich (hoffentlich) entsprechend verhalten.
READ_SYNC_STATS	<i>Synchronisierungsstatistiken lesen.</i> Beispielsweise den Verlauf bereits durchgeführter Synchronisationen einsehen.
REBOOT	Erlaubt den Neustart des Gerätes. Wie der Name es bereits andeutet: Diese App kann mal eben einen Reboot veranlassen.
RECEIVE_BOOT_COMPLETED	<i>Automatisch nach dem Booten starten.</i> App möchte benachrichtigt werden, wenn der Bootvorgang abgeschlossen ist. I.d.R. heißt das: Sie möchte nach dem Booten automatisch gestartet werden. Manchmal darf der Anwender dabei aber auch über die Konfiguration der App entscheiden, ob sie das dann wirklich tut.
RECEIVE_MMS, RECEIVE_SMS	<i>MMS empfangen, SMS empfangen.</i> Eingehenden MMS/SMS abfangen – da möchte wohl jemand mitlesen. Kann aber durchaus OK sein, wenn die App auf MMS/SMS reagieren soll. Auf der anderen Seite kann man damit eingehende Nachrichten auch "im Nirvana" verschwinden lassen...
RECORD_AUDIO	<i>Audio aufnehmen.</i> Tonaufnahmen erstellen. Das kann sowohl für ein "Diktaphon" genutzt werden – als auch zum Mitschneiden von Telefonaten.
RESTART_PACKAGES	<i>Anwendungen neu starten.</i> Laufende Apps neu starten. Wird z. B. verwendet, um von selbigen ein Backup erstellen zu können. (In der API-Referenz mittlerweile auf "deprecated" gesetzt, sollte also in neueren Versionen nicht mehr genutzt werden)
SEND_SMS	<i>Kurznachrichten senden.</i> Und zwar ohne Zutun des Benutzers, auch an richtig teure Premium-Dienste (womit klar ist, wozu "böse Apps" das gern hätten). Es gibt aber auch "gute" Gründe für diese Permission: Natürlich die SMS-Apps, aber teilweise auch In-App-Käufe, die nicht über Google Checkout abgewickelt werden.

Permission	Erklärung
SET_ACTIVITY_WATCHER	Die Ausführung von Systemaktivitäten beobachten. Wird meist für Debugging benutzt. Wenn es also keine Beta ist, hat der Entwickler vielleicht nur vergessen, das wieder raus zu nehmen.
SET_ALWAYS_FINISH	App kann sich selber <i>beenden</i> – also wirklich beenden, nicht nur in den Hintergrund gehen.
SET_ANIMATION_SCALE	<i>Allgemeine Animationsgeschwindigkeit einstellen.</i> Anpassung der Animationsgeschwindigkeit (schnellere/langsamere Animationen).
SET_PREFERRED_APPLICATIONS	App kann jeder Aktivität eine Default-App zuweisen (etwa den Browser zum Öffnen einer URL). In neueren Android-Versionen ohne Auswirkung.
STATUS_BAR	Kann die Status-Bar (Notification?) öffnen, schließen, und ausblenden. Meist will die App wohl letzteres, um einen "Vollbild-Modus" zu ermöglichen.
SUBSCRIBED_FEEDS_READ	<i>Abonnierte Feeds lesen.</i> Abrufen von Details zu den derzeit synchronisierten Feeds.
SUBSCRIBED_FEEDS_WRITE	<i>Abonnierte Feeds schreiben.</i> Änderungen an kürzlich synchronisierten Feeds vornehmen.
SYSTEM_ALERT_WINDOW	<i>Warnungen auf Systemebene anzeigen.</i> Fenster mit Systemwarnungen einblenden. Eine böswillige App kann so den gesamten Bildschirm blockieren. Sollte eigentlich nicht benutzt werden, da dies für System-Meldungen gedacht ist. Erlaubt das Anzeigen von "Alert Windows", d. h. Nachrichtenfenstern, die immer im Vordergrund angezeigt werden.
USE_CREDENTIALS	<i>Authentifizierungsinformationen eines Kontos verwenden.</i> Möchte die konfigurierten Zugangsdaten verwenden. Das heißt nicht unbedingt, dass es sie "zu sehen bekommt" – aber die App kann sich quasi "im Namen des Anwenders" anmelden.
USE_SIP	App kann Internet-Telefonie nutzen (SIP ist das gebräuchlichste Protokoll dafür)
VIBRATE	<i>Vibrationsalarm steuern.</i> Wird gern genutzt, um beispielsweise auf die Beendigung (oder auch den Start) einer Aktivität hinzuweisen – oder generell, um die Aufmerksamkeit des Anwenders zu wecken. Laute Töne sind ja nicht immer erwünscht. „Vibrieren“ heißt: Lass das Gerät zittern und summen...
WAKE_LOCK	<i>Standby-Modus deaktivieren.</i> App kann das System daran hindern, einen Ruhezustand einzunehmen (also den Bildschirm zu dimmen,

Permission	Erklärung
	die CPU "schlafen" zu lassen, etc.) Wäre doch blöd, wenn die Navi-App läuft und plötzlich der Bildschirm ausgeht.
WRITE_APN_SETTINGS	<i>Einstellungen für Zugriffspunktname schreiben.</i> App kann die Zugangsdaten zum Internet etc. (siehe APN) verändern. Meist geht es der App nur darum, den Namen des APN zu ändern – um die Verwendung des mobilen Datenverkehrs zu steuern (Beispiel: APNDroid).
WRITE_CALENDAR	<i>Kalenderdaten schreiben.</i> Diese Permission erlaubt lediglich den Schreib-, nicht aber den Lesezugriff auf den Kalender. Die damit versehene App kann also Termine hinzufügen, nicht aber lesen oder ändern.
WRITE_CONTACTS	<i>Kontaktdaten schreiben.</i> Wie WRITE_CALENDAR, nur in Bezug auf die Kontaktdaten bzw. Browser-History (Chronik) und Lesezeichen.
WRITE_HISTORY_BOOKMARKS	
WRITE_EXTERNAL_STORAGE	App darf beliebige Daten auf der (externen) SD-Karte lesen, schreiben, verändern und auch löschen – prinzipiell auch die Daten anderer Apps. Diese Permission ist aber beispielsweise essentiell für diverse Backup- und Kamera-Apps, die natürlich Daten auf der Karte manipulieren müssen.
WRITE_OWNER_DATA	<i>Eigentümerdaten schreiben.</i> Schreiben/verändern der auf dem Gerät gespeicherten Eigentümerdaten.
WRITE_SECURE_SETTINGS	<i>Allgemeine Systemeinstellungen ändern.</i> Lesen und Schreiben von Systemeinstellungen. "Secure Settings" können nur von Systemanwendungen (also solchen, die ins "ROM" integriert wurden) angefordert werden.
WRITE_SETTINGS	
WRITE_SYNC_SETTINGS	<i>Synchronisierungseinstellungen schreiben.</i> Schreibzugriff auf die Einstellungen der Synchronisation. Eine mit dieser Permission ausgestattete App kann u. a. die Synchronisation von Kontakten und Kalendern aktivieren bzw. deaktivieren.
com.android.vending.BILLING	In-App Payment (in der App integrierte Bezahldienste, die über den Google Playstore abgewickelt werden)

Weniger gebräuchliche sowie offensichtlich klingende Permissions (was heißt wohl [BRICK](#)? Ja, genau: Phone in Sachen Brauchbarkeit mit einem Ziegelstein vergleichbar zu machen, also unbrauchbar. Nicht lachen – diese Permission gibt es wirklich! So, jetzt lachen...) habe ich hier ausgelassen; die müssen also bei Bedarf unter eingangs genanntem Link selber nachgeschlagen werden. Oder man wirft einen Blick auf die App [AllPermissions](#) (bei AndroidPIT – aus dem Google Playstore wurde sie entfernt). Wie der Name bereits suggeriert, handelt es sich hier um

eine dummy-App, die alle (unter Android 2.1 verfügbaren) Permissions verlangt. Da AndroidPIT hier bei "Mouse Over" kurze Erklärungen anzeigt, lernt man dabei auch einiges.

Akku-Verbrauch im Mobilfunk-Standby mit Tasker bekämpfen

Unter [Was ist eigentlich "Mobilfunk-Standby", und warum braucht es so viel Akku?](#) wurden ja bereits die Hintergründe des *Mobilfunk-Standby* und seines Energieverbrauchs erklärt. Der Kernpunkt dabei war, dass bei schlechtem oder gänzlich abhanden gekommenem Funksignal zusätzliche Energie aufgewendet wird, um wieder eine gute Verbindung herzustellen. Was natürlich wenig Sinn macht, wenn man in einem Funkloch sitzt. In diesem Kapitel wollen wir nun betrachten, was sich dagegen unternehmen lässt.

Dazu habe ich mir zunächst einige Apps angeschaut, die das Problem bekämpfen wollen. Die Erfolge waren recht durchwachsen, doch zwei Apps schlugen sich recht gut. Zum Einen ist dies die rechts abgebildete App **Autopilot**, und zum zweiten **NoBars**, dessen Screenshot auf der linken Seite zu sehen ist.



Wie sich das auf die Akku-Statistiken auswirkt, zeigt der rechte Screenshot: Das *Telefonsignal* weist Löcher auf. Das sind die Zeiten, in denen sich das Gerät im Flugzeugmodus befand. Und demzufolge recht wenig Strom verbrauchte, was wiederum zu den längeren Akku-Laufzeiten führte.



Diese Apps (sowie die folgende Tasker-Lösung) habe ich jeweils unter vergleichbaren Bedingungen getestet. Ohne eine dieser Lösungen lag die Restladung meines Akkus nach ca. 11 Stunden bei etwa 60 Prozent. Sowohl *Autopilot* als auch *NoBars* schafften es, dies auf gut 70 Prozent zu verbessern. Meine Tasker-Lösung brachte es sogar auf fast 80 Prozent! Wie ist das zu schaffen?

Bei zu schwachem oder gänzlich abhanden gekommenen Signal (*Autopilot* erlaubt hier die Festlegung einer minimalen Signalstärke) versetzen die Apps den Androiden in den Flugzeugmodus. Nach einem festgelegten Zeitraum (bei *Autopilot* recht flexibel, bei *NoBars* hingegen nur grob einstellbar) wird der Flugzeugmodus wieder deaktiviert, sodass erneut die Signalstärke geprüft werden kann. Da im Flugzeugmodus das Radio-Modul deaktiviert ist, kann es auch keine (zusätzliche) Energie verbrauchen – so einfach ist das.



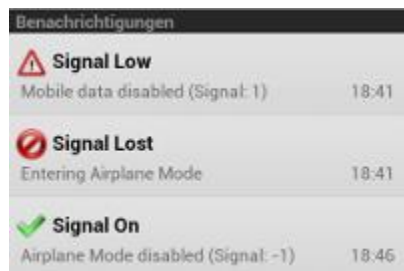
Welche der beiden Apps man nun einsetzen möchte, ist sicherlich eine Geschmacksfrage. Beide versahen ihren Dienst recht gut. *Autopilot* lässt sich umfangreicher konfigurieren, und bietet außerdem ein Protokoll; dafür schaltet es jedoch bei jeder Signalprüfung kurz den Bildschirm ein, was auf Dauer etwas nervt. *NoBars* gibt sich eher spartanisch, führt aber dennoch zu eben so guten Ergebnissen.

Und dann wäre da schließlich noch die App, die diesem Kapitel den Namen gab. In vielen Android-Foren werden Fragen nach dem Universum-und-so-weiter nämlich nicht mit 42, sondern lediglich mit 6 Buchstaben beantwortet: T-A-S-K-E-R. Eine 7-Tage-Testversion von *Tasker* findet sich im Playstore, die Vollversion kostet ca. 5 Euro – und ist jeden Cent absolut Wert.

Wie im rechten Screenshot zu sehen, habe ich mir also drei kleine Profile mit zugehörigen Tasks gebaut, die sich der Problematik *Mobilfunk-Standby* widmen sollen. Sie kommunizieren über eine gemeinsame Variable, die ich in meinem globalen *Init* profil initialisiere: In %SIGSTATE ist der aktuelle Zustand gespeichert. Dieser kann drei Werte annehmen: 0 (alles wunderbar), 1 (Signal relativ schwach) und 2 (Signal verschwunden).

Sofern %SIGSTATE kleiner als 1 ist, prüft *SigLowCheck* das Funksignal. Ist es unter einen eingestellten Wert (hier: 3) gefallen, werden die mobilen Daten deaktiviert (die würden jetzt nämlich auch zusätzliche Energie benötigen), und %SIGSTATE wird auf 1 gesetzt. Dieser Status bewirkt nun zweierlei: Zum Einen wird *SigLowCheck* nun nicht weiter ausgeführt (Bedingung war ja: kleiner als 1), zum Anderen schlägt nun sofort *SigReturnCheck* zu, welches nur bei einem Wert größer als 1 tätig wird.

SigReturnCheck setzt zunächst einen Timer, und wird dann erst 5 Minuten später tätig. Es aktiviert die mobilen Daten wieder, und deaktiviert den Flugzeugmodus. Dann setzt es erneut einen Timer, und 15 Sekunden später die Variable %SIGSTATE auf 0 – womit es sich selbst deaktiviert, und den anderen beiden Kandidaten den Weg frei macht.



Der Dritte im Bunde ist *SigLostCheck*, und prüft (sofern %SIGSTATE kleiner ist als 2) das Funksignal. Fällt es unter den konfigurierten Schwellwert (2), wird in den Flugzeugmodus gewechselt, und %SIGSTATE auf 2 gesetzt. Ein eventuell aktiviertes WLAN bleibt dabei auch aktiviert.

Um den Anwender auf dem Laufenden zu halten, blenden alle drei entsprechende Informationen im Nachrichtenbereich ein (siehe linkes Bild). Der jeweils aktuelle Zustand steht dabei zuunterst, und das Neuauftreten eines Zustands (etwa der erneute Wechsel in den Flugzeugmodus) überschreibt den bereits vorhandenen Eintrag. War noch kein Eintrag vorhanden, macht sich auch der Vibrator kurz bemerkbar.



Klingt doch ganz praktikabel. Und zum Nachbauen, hier die einzelnen Tasks sowie ihre zugehörigen Profile:

Task "InitVars":

- Variable → Variable Set: %SIGSTATE To 0

Task "SigLow":

- Net → Mobile Data: Off
- Variable → Variable Set: %SIGSTATE To 1
- Alert → Notify Vibrate:
 - Title: "Signal Low"
 - Text: "Mobile data disabled (Signal: %CELLSIG)"

Task "SigLost":

- Variable → Variable Set: %WLANSTATE To 0
- Variable → Variable Set: %WLANSTATE To 1 IF %Wifi ~ on
- Net → Airplane Mode: On
- Net → Wifi: On IF %WLANSTATE ~ 1
- Variable → Variable Set: %SIGSTATE To 2
- Alert → Notify Vibrate:
 - Title: "Signal Lost"
 - Text: "Entering Airplane Mode"

Task "SigReturn":

- Task → Wait: 5 Minutes
- Net → Airplane Mode: Off
- Task → Wait: 15 Seconds (dem Androiden Zeit geben, ein neues Signal zu finden!)
- Net → Mobile Data: On IF %ROAM ~ Off (damit es nach dem Urlaub im Ausland keine böse Überraschung gibt: Nur wieder aktivieren, wenn nicht im Roaming!)
- Variable → Variable Set: %SIGSTATE To 0
- Alert → Notify Vibrate:
 - Title: "Signal On"
 - Text: "Airplane Mode disabled (Signal: %CELLSIG)"

Profil SigLowCheck:

- State → Variable → Variable Value:
 - Name: "%SIGSTATE"
 - Op "Math: Less Than"
 - Value "1"
- State → Phone → Signal Strength: From 0 To 2
- Task: SigLow

Profil SigLostCheck:

- State → Variable → Variable Value:
 - Name: "%SIGSTATE"
 - Op "Math: Less Than"
 - Value "2"
- State → Phone → Signal Strength: From 0 To 1
- Task: SigLost

Profile SigReturnCheck:

- State → Variable → Variable Value:

- Name: "%SIGSTATE"
 - Op "Math: Greater Than"
 - Value "0"
- Task: SigReturn

Profile Init:

- Event → Tasker → Monitor Start
- Task: InitVars

Wer mag, kann sich natürlich nach Belieben weiter austoben: Sound abspielen lassen, das Display zum Blinken bringen... Aber die essentiellen Dinge zum Stromsparen sind alle bereits enthalten, einschließlich der Informationen zum aktuellen Status.

Diebstahl-Schutz mit Tasker

Schrieb ich doch im Kapitel [Diebstahlschutz](#), ein solcher ließe sich eventuell auch mit Tasker realisieren? In der Tat ist dies möglich. Eine Variante des "Device-Trackings" (aka "Wo ist mein Androide?") stellt [Kevin Purdy](#) in einem [Artikel auf Lifehacker.COM](#) mit einer bebilderten Schritt-für-Schritt-Anweisung vor. Zwar auf Englisch – aber dank der guten Bebilderung sollten schon geringfügige Sprachkenntnisse ausreichend sein, um dem How-To folgen zu können.

Damit wäre die Lokalisierung eines abhanden gekommenen Androiden abgedeckt. Damit dies nicht in der Absicht eines "bösen Menschen" geschieht (sondern allenfalls wir das Gerät "irgendwo liegen gelassen haben könnten"), hier noch ein kleiner Anreiz zur Umsetzung eines "Diebstahl-Alarms". Selbiger soll losgehen, wenn ein Unbefugter sich unseres Androidens bemächtigt – nicht aber, wenn wir selbst ihn in die Hand nehmen. Dazu könnte wie folgt vorgegangen werden:

1. **Profil 1: Bildschirm wurde entsperrt**
 - Bedingung: Event → Display → Display unlocked
 - Task: Variable → Variable Set, U_UNLOCKING = 1
2. **Profil 2: Alarm deaktivieren**
 - Bedingung 1: Event → Hardware → Button: Camera
 - Bedingung 2: State → Variable Value U_UNLOCKING > 0
 - Task: Variable → Variable Set, U_UNLOCKING = 0
3. **Profil 3: Alarm-Count-Down auslösen**
 - Bedingung: State → Variable Value U_UNLOCKING > 0
 - Task: XXX

Bei "XXX" ist natürlich die entsprechende Aktion einzusetzen. Beispielsweise mit einer While-Schleife 10 Sekunden warten, ob doch noch entschärft wird – und andernfalls anschließend ein entsprechendes MP3 abspielen, welches laut "DIEBE! ICH BIN EIN GEKLAUTES TELEFON!" brüllt, und dabei Polizeisirene, "Roten Alarm" der Enterprise, o. ä. einblendet.

Wie ist das Ganze gedacht? Über eine Variable (U_UNLOCKING) schauen wir, ob überhaupt etwas passieren muss (ist sie ungleich 0?). Dazu wird sie bei Entsperrung des Bildschirms (ggf. durch andere Events wie "Gerät bewegt" ersetzen) auf 1 gesetzt – dies erledigt "Profil 1".

Wird nun beim/kurz nach dem Entsperrten des Bildschirms der Kamera-Button gedrückt, gibt "Profil 2" Entwarnung (setzt U_UNLOCKING=0; diese Aktion kann natürlich gegen einen beliebigen anderen Event ausgetauscht werden: Etwa Umdrehen oder Schütteln des Androiden). Passiert dies nicht, kümmert sich "Profil 3" um die entsprechende Alarm-Aktion.

Dies ist ein rein theoretischer Ansatz – sollte das jemand einmal umgesetzt haben, freue ich mich auf entsprechendes Feedback!

P.S.: Sollte das Gerät sich dennoch unerlaubt entfernt haben, so zeigt ein [Video-Tutorial](#) bei Youtube, wie sich auch die Ortung desselben mittels *Tasker* realisieren lässt...

Secret Codes oder Magische Nummern

Klar kann man mit einem Telefon telefonieren. Dazu gibt man eine Ziffernfolge ein, und drückt die Taste für "Abheben". Was aber passiert, wenn man noch ein paar Sonderzeichen hinzufügt?

Disclaimer: Es kann sein, dass diese Codes nicht auf allen Geräten funktionieren. Sofern sie nicht im Handbuch des Gerätes aufgeführt sind, mag das durchaus seine Gründe haben: So kann der eine oder andere Punkt, unsachgemäß angewendet, u. U. Schäden verursachen. Ich übernehme keinerlei Garantien dafür, dass folgende Codes a) funktionieren, b) das tun, was da steht, oder c) "folgenfrei" genutzt werden können. Insbesondere übernehme ich keinerlei Verantwortung für etwaige negative Folgen! Für etwaige positive Folgen stelle ich natürlich gern mein Bankkonto zur Verfügung...

Es kann sein, dass nach Eingabe des einen oder anderen Codes nichts passiert. Es kann aber auch sein, dass dies nur so scheint (bei meinen Tests fand ich anschließend – am nächsten Tag – einige der "magischen Nummern" in der "Verbraucherliste" wieder). Es können hier durchaus Hintergrund-Dienste gestartet oder aber "versteckte Klassen/Menüs" in Apps freigeschaltet werden...

Code	Bedeutung
*##*0*##*	Test des LCD-Displays
*#0011#	GSM-Infos
*#0228#	ausführliche Infos zum Barreriestatus
*##*0283*##*	Loopback-Test
**05*<PUK Code>*<neue PIN>*<neue PIN zur Bestätigung>#	Entsperren des Telefons aus dem Notruf-Modus
*##*0588*##*	Test des Annäherungs-Sensors
*#06#	IMEI
*##*0673*##*	Einer der beiden Codes führt zu einem Audio-Test
*##*0289*##*	
*#0782#	RTC (Real-Time-Clock) auslesen und anzeigen
*##*0842*##*	Test für Vibrator (oh ja!) und Hintergrund-Beleuchtung
*##*1111*##*	FTA Software Version
*##*1234*##*	Firmware-Info
*135#	Eigene Rufnummer anzeigen
*##*1472365*##*	ein kurzer GPS-Test (und Einstellungen)
*##*1575*##*	ein weiterer GPS-Test
*##*197328640*##*	Service-Menü mit verschiedenen Test-Möglichkeiten
*#21#	Status der Anrufweiterleitungen überprüfen
*##*2222*##*	FTA Hardware Version
*#2263#	Bandnutzung
*##*232331*##*	Bluetooth-Test
*##*232337*##*	MAC-Adresse des Bluetooth-Interfaces
*##*232338*##*	MAC-Adresse des WLAN-Interfaces anzeigen

Code	Bedeutung
#232339 ***#526*** ***#528***	Einer dieser Codes führt zu den WLAN-Tests...
#2432546 (*##CHECKIN##*)	Nach OTA -Updates suchen
#2663	Touch-Screen Version anzeigen
#2664	Touch-Screen Test
#273283*255*663282	angeblich für eine schnelle Sicherung der Medien-Dateien (Fotos, Videos..) gut. Von wo und nach wo? Habe ich nicht probiert...
*2767*3855#	Factory Format (Wipe). Vorsicht damit!!!
#31#<Rufnummer>	Mit unterdrückter Rufnummer anrufen
#3264	RAM Test / RAM-Version anzeigen
#3424	HTC Function Test
#34971539	Kamera-Menü mit folgenden Punkten: Update mit Firmware aus Bild (keinesfalls! Sonst futsch!); Update mit Firmware von SD-Karte; Versions-Info; Update-Counter
*43# / #43# / *#43#	Anklopfen aktivieren / deaktivieren / Status abfragen
#44336	Herstellungszeitpunkt und Laufende Nummer
#4636 (*##INFO##*)	Öffnet ein Menü, aus dem sich wählen lässt: Telefon-Infos, Akku-Infos, Akku-History, Verbrauchsstatistiken. Der Umfang kann je nach Android-Version unterschiedlich ausfallen.
#4646#	Feldtest (Details zu Akkustand + Funknetz, Wechsel UMTS/GSM)
#4986*2650468	Diverse Hardware-Infos
#564	QXDM (Qualcomm eXtensible Diagnostic Monitor) Logging FrontEnd
#7262626	Ein Bett im Kornfeld... äh, Feld-Test
#7269	Standard Device-Logging (Device [logcat?], AT-Befehle, Kernel [dmesg?] -- HTC?)
*#7465625#	Netz- und Subnetzsperrung, Simlock, Service Provider und Corporate Lock (Galaxy-S?)
#7594	Verhalten des Einschalt-Knopfes ändern (z. B. direktes Abschalten ohne Menü)
#7780	Zurücksetzen auf Werkseinstellungen
#8255	GoogleTalk Service Überwachung
#8350	Logging der Anrufe deaktivieren
#8351	Logging der Anrufe aktivieren
*#9090#	Service-Modus UART/USB
#9696	FTP Test
*N# (TCom: *T#)	Wird eine SMS mit dieser Zeichenfolge begonnen, erfolgt eine SMS Empfangsbestätigung

Weitere spezielle magische Nummern gibt es für Geräte von...

- ...[LG](#)
- ...[Motorola](#)
- ...[Samsung](#)
- ...[Samsung](#) (SGS)
- ...[Samsung](#) (SamFirmware)

Leistungsaufnahme verschiedener Komponenten

Heise hat für seinen Artikel [Energiesparplan](#) ein Motorola Milestone angepasst und ausführlich getestet, was welche Komponente so verbraucht. Und wie man das Einschränken kann (das war jetzt eine klare Empfehlung, den Artikel zu lesen!). Die Daten davon werden einerseits ein "war ja klar", aber bei einigen Daten auch ein "oh, das hätte ich jetzt nicht gedacht" hervorrufen. Passende Angaben zum Galaxy S3 hat die c't in ihrer [Ausgabe vom August 2012](#) gesammelt – und dabei den Test gleich noch ein wenig ausgeweitet.

Anmerkungen zum Galaxy S3 (*): Der Energieverbrauch des Displays hängt hier auch stark von den dargestellten Inhalten ab, was auf das verwendete AMOLED-Display zurückzuführen ist: Bei vollständig schwarzem Display entspricht der Verbrauch auf allen Stufen nicht mehr als das Minimum.

Weiterhin sollte man beim Vergleich der Werte auch die unterschiedliche Hardware-Ausstattung im Hinterkopf behalten: So werkelt beispielsweise im Galaxy S3 ein mit 1,4 GHz getakteter Quad-Core Prozessor, und zur Anzeige dient ein 4,8 Zoll Display – im Milestone waren es noch eine Single-Core CPU mit 550 MHz und ein 3,7 Zoll Display.

Betriebszustand	zusätzliche Leistungsaufnahme	
	Motorola Milestone	Samsung Galaxy S3
Videoaufnahme ¹	1557 mW	1683 mW
UMTS Upload	1410 mW	1033 mW
UMTS Download	1349 mW	1074 mW
EDGE Upload	1179 mW	
WLAN Download	1158 mW	549 mW
Video abspielen (fullscreen) ¹	1135 mW	597 mW
UMTS-Telefonat	983 mW	637 mW
Kamera ¹	934 mW	1460 mW
EDGE Download	853 mW	
Bluetooth empfangen	751 mW	487 mW
Display (höchste Stufe)	730 mW	1568 mW*
GPS Suche	550 mW	263 mW
GSM-Telefonat	511 mW	297 mW
Bluetooth senden	487 mW	454 mW
WLAN Upload	479 mW	488 mW
Display (niedrigste Stufe)	310 mW	567 mW
WLAN Tether 2		372 mW
MP3 abspielen über Bluetooth		296 mW
MP3 abspielen	160 mW	153 mW
UMTS Standby	18,3 mW	13,8 mW
GSM/EDGE Standby	11,6 mW	9,5 mW
WLAN Standby 2,4 GHz	7,8 mW	9,3 mW
WLAN Standby 5 GHz	N/A	14,6 mW
NFC Standby	N/A	4 mW
Bluetooth Standby	2,8 mW	1,8 mW
GPS Standby	0,4 mW	0,7 mW
WLAN Tether Download ³		1254 mW

- 1 Leistungsaufnahme des Displays bereits abgezogen
- 2 Tethering aktiv, 1 Benutzer
- 3 Download vom Notebook per WLAN-Tether

Für die Gesamt-Leistungsaufnahme muss natürlich noch die Grundlast hinzugerechnet werden (was das Gerät verbraucht, wenn alles in der Tabelle genannte abgeschaltet ist; also "Flugmodus"). Das wären ganze fette 6,4mW. Wird das Gerät also des Nachts in diesen versetzt, spart das bereits enorm:

Betriebszustand	zusätzliche Leistungsaufnahme	
	Motorola Milestone	Samsung Galaxy S3
Flugmodus	6,4 mW	6,4 mW
GSM Bereitschaft	18 mW	9,5 mW
GSM Bereitschaft + WLAN Standby	25,8 mW	18,8 mW
GSM Bereitschaft + WLAN Standby + Bluetooth Standby	28,6 mW	20,6 mW
UMTS Bereitschaft	24,7 mW	10,9 mW
UMTS Bereitschaft + mobile Daten aktiv		13,8 mW
UMTS Bereitschaft + WLAN Standby	32,5 mW	20,2 mW
UMTS Bereitschaft + WLAN Standby + Bluetooth Standby	35,3 mW	22,0 mW

Die Werte sind natürlich alle spezifisch für o.g. Motorola Milestone bzw. das Samsung Galaxy S3, und können auf anderen Geräten abweichen. Die Größenordnungen sollten aber zumindest ähnlich sein.

Nur so nochmal gesagt: Selbst wenn WLAN etc. alles aus sind, und nur Telefonate (und SMS) noch durchkommen (den Datenverkehr also mal ganz außen vorgelassen), reduziert sich der Verbrauch im Flugmodus auf ein Drittel (GSM) oder gar ein Viertel (UMTS). Einmal 6 Stunden angenommen (von 0 Uhr bis 6 Uhr), hält der Akku damit (theoretisch) für bis zu gut 2 Stunden länger. Natürlich wieder nur, wenn man dann anschließend auch nichts damit macht – also wieder so ein "Laborwert". Trotzdem kann man sich leicht ausrechnen: Sechs Stunden Flugmodus (statt GSM Bereitschaft) schaffen Raum für zusätzliche ca. 8 Minuten GSM Telefonat...