Seminar:

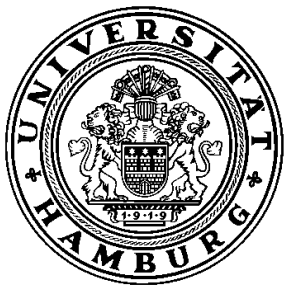
„Android: Plattform für mobile Geräte“

Publishing of Android Applications

von
Artiom Wulis

Wintersemester 2009/2010

Betreuer:
Julian Kunkel



Universität Hamburg
Department Informatik
MIN Fakultät
AB Wissenschaftliches Rechnen

Inhaltsverzeichnis

1 Einleitung

Mit der zunehmender Beliebtheit von Android Betriebssystem, wächst auch die „Android-Gemeinschaft“ und somit auch die Anzahl potentieller Entwickler/innen. Eine erste Anwendung ist schnell geschrieben. Will man später diese auf dem Endgerät übertragen, oder sogar kommerziell auf Android Market vermarkten, so müssen einige wichtige Vorkehrungen gemacht werden.

Diese Seminararbeit kann als Einführung und Anleitung für die Veröffentlichung von Android Anwendungen auf den Endgeräten und Android Market betrachtet werden.

Da Android Anwendungen in JAVA geschrieben werden, wird hier Eclipse (IDE), die sich als Standard-Etwicklungsumgebung für JAVA etabliert hat, verwendet. Daher werden zumindest grundlegende Eclipse Kenntnisse vorausgesetzt.

2 Publishing Voraussetzungen

Bevor man seine Anwendung veröffentlichen kann, müssen bestimmte Voraussetzungen erfüllt werden. Nachfolgende Liste, stellt grob die Einzelne Schritte dar, die im weiteren Textverlauf näher erläutert werden. Es müssen jedoch nicht alle Punkte der Liste, wie z.B. Testen, abgearbeitet werden, es ist aber empfehlenswert alles zu machen, zumal Testen der Anwendungen zu jedem guten Programmierstil dazu gehört.

2.1 Publishing Checkliste:

1. Anwendung ausgiebig Testen
2. Endbenutzer-Lizenzvereinbarung hinzufügen
3. Icon und Label spezifizieren
4. Logger und Debugger ausschalten, temporäre. Dateien löschen
5. Anwendung versionieren
6. Anwendung kompilieren und signieren

Im weiteren Verlauf wird Eclipse mit ADT Plugin (Android Development Tools) und Android SDK benutzt. Eine Installationsanleitung der beiden Komponenten findet man auf der Android Developerhomepage[1].

1 http://developer.android.com/sdk/installing.html

2.1.1 Anwendung ausgiebig Testen

Neben den Unit Test, ist es empfehlenswert die Android Anwendung auf möglichst vielen Endgeräten zu testeten.

Bei der Vielfallt der Android Geräten, ist logisch, dass man nicht alle Geräte besitzt. Daher gibt es vorgefertigte Android Virtual Devices (AVD). Eigene AVD‘s können selbstverständlich auch erstellt werden. Dabei kann man zwischen den verschiedenen Android OS Versionen auswählen.

Seltene oder nicht vorhandene Geräte, können mit dem Emulator weitestgehend emuliert werden. Es stehen unter anderem folgende Startparameter: -dpi, -netspeed, -cpu-delay, zur Auswahl.
Emulator Startparameter findet man unter Eclipse: Window > Preferences > Launch > Default emulator options.

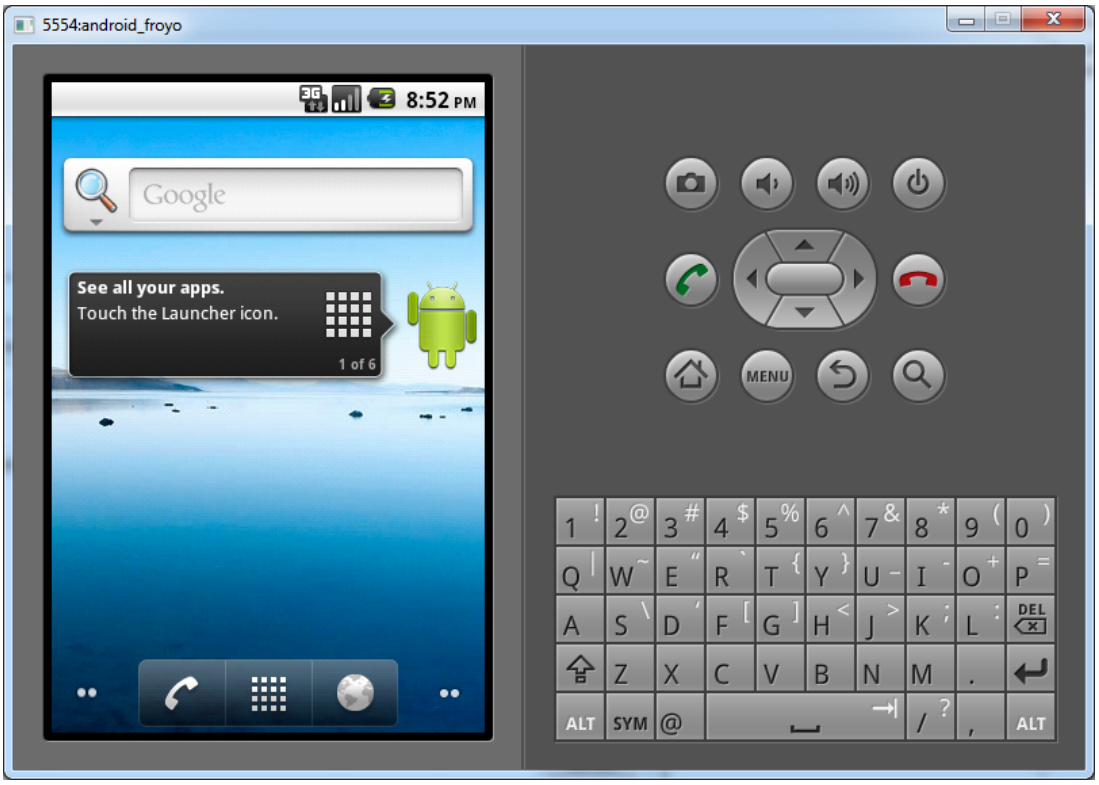


Abbildung 1. Android AVD ohne Skin mit Android OS. 2.2

Eclipse mit ADT Plugin bietet zwar einen mächtigen Android Emulator, doch dieser kann ein echtes Android Gerät nicht ersetzen. Physische Eigenschaften wie Netzqualität, Bildschirmbedienung, Akkuverbrauch und vieles mehr, sind von Gerät zu Gerät sehr unterschiedlich.

2.1.2 Endbenutzer-Lizenzvereinbarung hinzufügen

Für die Veröffentlichung auf Android Market ist eine Endbenutzer-Lizenzvereinbarung nicht nötig. Will man aber eine EULA haben, so existieren im Internet viele Anleitungen mit Beispielen, wie eine EULA auf Android OS realisiert werden könnte.

2.1.3 Icon und Label spezifizieren

Jede Anwendung braucht sowohl einen Namen, als auch ein Logo. Der Name und das Logo sind überall, wie z.B. auf Android Market, Programmanager und Homescreen sichtbar.

Name *android:label="@string/app_name"* und Logo *android:icon="@drawable/icon"* werden in der *AndroidManifest.xml* unter application definiert.

```
<application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
</application>
```

Auszug aus AndroidManifest.xml

Während der Entwicklung mit Eclipse, gibt es im typischen Android Projektverzeichnis den *res* Ordner. Dieser hat unter anderem diese Ordner*: drawable-hdpi*, *drawable-mdpi*, *drawable-ldpi*, *values*. Die drei drawable Ordner beinhalten in unserem Beispiel eine Datei *icon.png*. Die *drawable* Ordner haben drei Endungen: *hdpi* (hohe Auflösung), *mdpi* (mittlere Auflösung) und *ldpi* (kleine Auflösung). Dementsprechend muss die Größe der Bilddatei angepasst werden. Der Ordner *values* beinhaltet die Datei *strings.xml* in der die Variable *app_name* deklariert ist.

```
<resources>
        <string name="app_name">Hello, Android</string>
</resources>
```

Auszug aus strings.xml

2.1.4 Logger und Debugger

Logger und Debugger sind in der Entwicklungsphase unverzichtbar. Diese, sowie evtl. temporäre Dateien müssen vor der Veröffentlichung deaktiviert und gelöscht werden.

Debugger wird in der AndroidManifest.xml ausgeschaltet.

```
<application
        android:debuggable="false"
</application>
```

Für diejenigen, die den puren Quelltext von AndroidManifest.xml fürchten und lieber mit einem Grafischen Editor arbeiten, bietet Eclipse mit Android ADT eine grafische Oberfläche, in der man bequem alles verwalten kann. Im folgenden Textverlauf werden sowohl Ausschnitte aus AndroidManifest.xml, als auch Screenshots von Eclipse zu sehen sein.

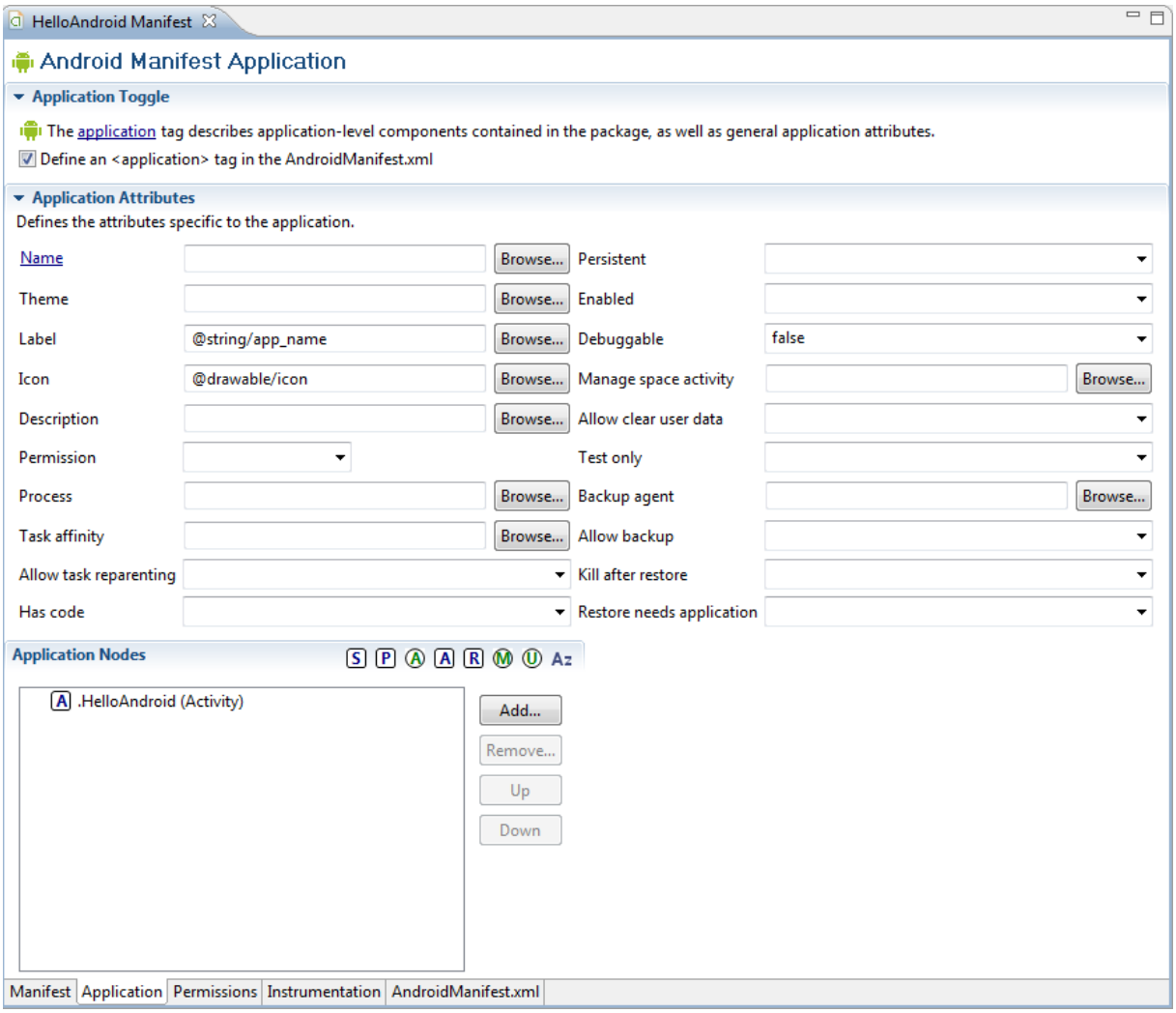


Abbildung 2. AndroidManifest.xml (Application Ansicht) in der Grafischen Oberfläche

2.1.5 Anwendung versionieren

Bevor wir zum eigentlichen Versionieren kommen, stellt sich die Frage wozu müssen wir unsere Anwendung überhaupt versionieren.

Endbenutzer müssen die Möglichkeit haben, die Programmversion zu sehen. Diese Möglichkeit ist sehr sinnvoll, wenn man ein Update / Upgrade installieren möchte oder in einem Supportfall als Entwickler Hilfestellung leisten will.

Fremde und auch eigene Anwendungen müssen ebenfalls die Versionsangaben von installierten Anwendungen sehen können. So kann Inkompatibilität vermieden werden.

Will man die Anwendung auf Android Market oder ähnlichen Seiten veröffentlichen, so muss man ebenfalls Versionsangaben machen.

Wie bereits erwähnt, müssen einigen Versionsangaben gemacht werden bevor die fertige Android Anwendung kompiliert wird. In der Android Manifest gibt es fünf Attributen.

Zwei davon sind *android:versionCode* und *android:versionName*.

Der *android:versionCode* ist ein Integer Wert und ist somit für andere Anwendungen und Systeme evaluierbar. Dabei gibt es keine Unterscheidung zwischen kleineren oder größeren Updates. Nach jedem Update wird der Wert um eins erhöht. Dieser Wert sollte für Endanwender verborgen bleiben.

Für die Endanwender ist *android:versionName* da. Dieser Wert ist ein String und ist somit für Anwendungen nicht evaluierbar. Dabei ist eine Unterscheidung zwischen kleineren und größeren Updates möglich. Gültige Werte von *android:versionName* sind z.B. 1.21 / 3a. Dementsprechend sollte dieser Wert für die Endanwender sichtbar sein.

Man sollte immer die Werte für *android:versionCode*nd und *android:versionName* festlegen und nach jedem Release auch beide erhöhen.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.package.name"
    android:versionCode="2"
    android:versionName="1.1">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    ...
  </application>
</manifest>
```

Auszug aus AndroidManifest.xml

Android Os prüft nicht diese Versionsangaben. Es werden jedoch die SDK Versionsangaben bei der Installation geprüft.

Benötigt die eigene Anwendung eine bestimmte Android Plattform Version oder ist für eine bestimmte Version optimiert, so kann man das mit der der Android SDK Version festlegen.

Dafür sind die restlichen drei Attribute da, nämlich *android:minSdkVersion, android:targetSdkVersion* und *android:maxSdkVersion*.

Wie der Name schon sagt, gibt die *mindSdkVersion* die minimale Android Plattform Version an, welche die Anwendung braucht. Benötigt eine Android Anwendung z.B. Flashfunktionalität, welche erst ab Android 2.2 (Api Level 8), so muss die *minSdkVersion* den Wert 8 haben.

Die *targetSdkVersion* gibt an, für welche Android Version die Anwendung programmiert und optimiert ist. Es bedeutet jedoch nicht, dass die Anwendung auf früheren Versionen nicht läuft sondern, dass eventuelle Funktionalitäten, die mit der *TargetSdkVersion* erschienen sind, in den früheren Versionen nicht genutzt werden können. Die *TargetSdkVersion*, ersetzt jedoch nicht die *minSdkVersion*.

Vollständigkeitshalber sollte die *maxSdkVersion* erwähnt werden, diese Angabe findet in der Praxis jedoch keinen Gebrauch. Die *maxSdkVersion* stellt die Obergrenze da, also bis zu welcher Android Version die Anwendung laufen darf.

Folgendes Szenario verdeutlicht die Problematik mit *maxSdkVersion*.

Auf dem Endgerät mit Android Version 2.1 (Api Level 7) ist eine Anwendung installiert welche als *maxSdkVersion* den Wert 7 hat. Nun kommt ein Update auf Android Version 2.2 (Api Level 8) heraus. Nach der Installation des Updates funktioniert das zuvor installierte Programm nicht mehr und wird gelöscht.

Da alle zukünftigen Android Versionen mit den früheren Versionen kompatibel sind, macht es keinen Sinn die *maxSdkVersion* zu benutzen. Das empfehlen auch die Android Entwickler auf der Entwicklerhomepage.

Die Prüfung der maxSdkVersion wird ab Android 2.0.1 nicht mehr gemacht.

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
   <application …</application>
   <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="7"/>
</manifest>
```

Auszug aus AndroidManifest.xml

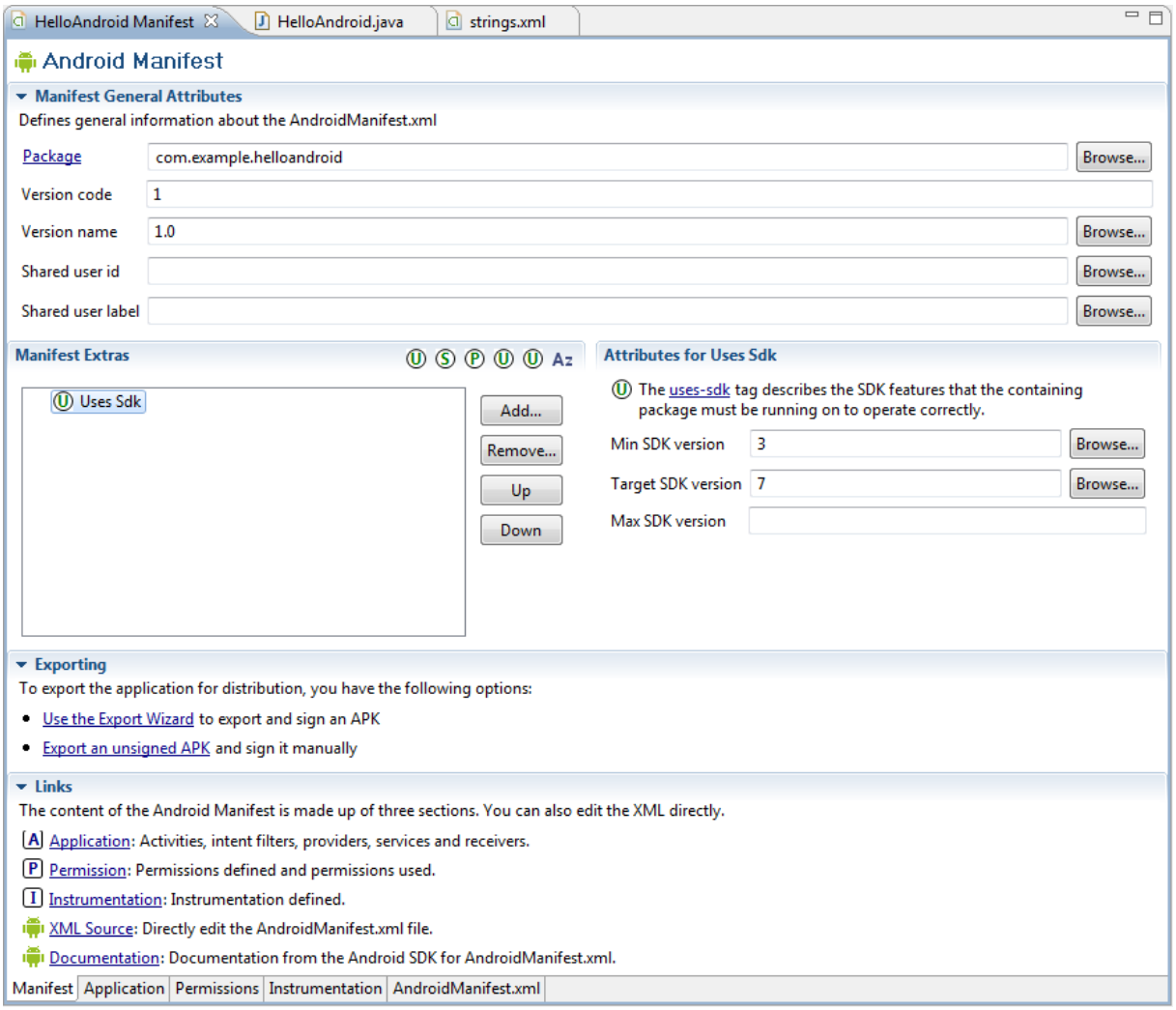


Abbildung 3. AndroidManifest.xml (Manifest Ansicht) in der Grafischen Oberfläche

2.1.6 Anwendung signieren

Bevor wir zum eigentlichen Signieren kommen, sollte noch erwähnt werden, dass alle Android Anwendungen signiert sein müssen. Dabei kann man glücklicherweise eigene Zertifikate zum Signieren benutzen und ist somit auf teurere Zertifizierungsstellen nicht angewiesen. Selbstverständig kann man die kostenpflichtigen Zertifizierungsstellen auch benutzen. Android Os prüft beim Installieren, ob die Anwendung signiert ist. Das dient der Sicherheit und soll das Vertrauensverhältnis zwischen den Entwickler und Endanwender stärken. So kann man als Endanwender selber prüfen, ob die Anwendung wirklich von dem Hersteller kommt, oder es sich um einen Trojaner handelt.

Es gibt zwei wichtige Aspekte, die bei der Signierung bedacht werden müssen.

Zum einem ist es empfehlenswert alle Anwendungen mit demselben Zertifikat zu signieren. Mochte man in Zukunft Updates für die eigene Anwendung herausbringen, so muss man das Update ebenfalls mit demselben Zertifikat wie die Hauptanwendung signieren. Beim Installieren prüft das Android Os nach, ob die Zertifikate übereinstimmen. Sollte das nicht der Fall sein, wird Android Os sofern möglich, das Update als eigenständiges Programm und nicht als Update installieren. Dies ist jedoch nicht der Sinn eines Updates.

Android Os erlaubt den Anwendungen, die mit demselben Zertifikat signiert sind, in demselben Prozess zu laufen. Dabei werden die Anwendungen vom System wie eine Anwendung behandelt. Diese Möglichkeit erlaubt Anwendungen wie z.B. Office Programme, modular zu gestallten. Dadurch kann man einige Module z.B. erweitern, updaten und löschen.

Ein weiterer Vorteil besteht darin, dass das Android Os „Signaturbasierte Berechtigungen“ unterstützt. Programme mit demselben Zertifikat können Funktionalitäten des anderen benutzen. Ferner ist es auch möglich, auf sichere Art Daten untereinander auszutauschen.

Zum anderem ist es ebenfalls empfehlenswert die Gültigkeitsdauer des Zertifikates vorausschauend zu vergeben.

Wie oben schon beschrieben, ist es in der Praxis üblich Updates oder Upgrades herauszubringen. Dabei darf das Zertifikat nicht ablaufen, bevor das eigentliche Programm noch vom Hersteller noch unterstützt und erweitert wird. Hat man mehrere Anwendungen, so muss das Zertifikat so lange gültig sein, bis die letzte Anwendung noch vom Hersteller unterstützt wird.

Möchte man die eigene Anwendung auf Android Market veröffentlichen, so muss das Zertifikat frühestens nach 22.10.2033 enden.

Debug mode

Während der Entwicklung und Testphase braucht man sich nicht darum zu kümmern, wie die Anwendungen signiert werden. Das Signieren übernimmt das Eclipse. Dabei wird die Anwendung in dem *Debug mode* kompiliert und mit dem Debugzertifikat signiert. Dieses Debugzertifikat ist 365 Tage gültig. Sollte die Entwicklungsphase über ein Jahr dauern, so muss dann der Schlüssel oder der ganze Keystore gelöscht werden. Danach erzeugt Eclipse wieder einen Debugzertifikat, der ein Jahr gültig wird.

Ein Keystore ist ein Zertifikatspeicher, der verschiedene Schlüssel zum Ver- und Endschlüsseln speichert. Die Zertifikate werden durch den eindeutigen Aliasnamen unterschieden. Wenn eine Anwendung in *Debug Mode* kompiliert wird, werden immer wieder die gleichen Daten für das Signieren benutzt.

Keystore name:	*"debug.keystore"*
Keystore password:	*"android"*
Key alias:	*"androiddebugkey"*
Key password:	*"android"*
CN:	*"CN=Android Debug,O=Android,C=US"*

Debug Keystore und Schlüsseldaten

Release mode

Möchte man seine Anwendung auf viele Android Geräte installieren oder im Android Market veröffentlichen, so muss die Anwendung im *Release mode* kompiliert werden. Anwendungen die mit dem Debugzertifikat signiert worden sind, lassen sich nur auf den Geräten installieren, die an dem Computer angeschlossen sind. Ferner müssen die Geräte sich in dem Debugmodus befinden. Im *Release mode* kompilierte Anwendungen sind standardmäßig nicht signiert. Deshalb müssen dieser erst mal mit dem privaten Schlüssel signiert werden.

Wenn noch kein privater Schlüssel vorhanden ist, so lässt sich dieser mit dem Tool *Keytool.exe* generieren. *Keytool.exe* liegt on JavaSDK/bin Ordner. Weitere Informationen zu Privaten Schlüssel und Keyool.exe findet man auf der Java Homepage[2].

Führt man keytool.exe mit dem Parameter

```
keytool.exe -genkey -v -keystore my-release-key.keystore -alias alias_name
-keyalg RSA -validity 10000
```

passiert folgendes:

Es wird ein Schlüsselpaar bestehend aus privaten und öffentlichen Schlüssel von Typ RSA 1024Bit (-keyalg *RSA*) erzeugt (-genkey). Dabei wird mit dem öffentlichen Schlüssel ein selbstsigniertes Zertifikat mit einer Gültigkeit von 10000 Tagen erzeugt. Der private Schlüssel wird als „alias_name“ in dem neu erstellten Keystore namens *my-release-key.keystore* gespeichert.

2 http://download.oracle.com/javase/1.5.0/docs/tooldocs/#security

PS C:\Program Files\Java\jdk1.6.0_18\bin> keytool.exe -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -validity 10000
Geben Sie das Keystore-Passwort ein:
Geben Sie das Passwort erneut ein:
Wie lautet Ihr Vor- und Nachname? [Unknown]: Heinz Mustermann
Wie lautet der Name Ihrer organisatorischen Einheit? [Unknown]: Android developer
Wie lautet der Name Ihrer Organisation? [Unknown]: Uni HH
Wie lautet der Name Ihrer Stadt oder Gemeinde? [Unknown]: Hamburg
Wie lautet der Name Ihres Bundeslandes oder Ihrer Provinz? [Unknown]: Hamburg
Wie lautet der Landescode (zwei Buchstaben) für diese Einheit? [Unknown]: DE
Ist CN=Heinz Mustermann, OU=Android developer, O=Uni HH, L=Hamburg, ST=Hamburg, C=DE richtig? [Nein]: ja
Erstellen von Schlüsselpaar (Typ RSA, 1.024 Bit) und selbstunterzeichnetem Zertifikat (SHA1withRSA) mit einer Gültigkeit von 10.000 Tagen
für: CN=Heinz Mustermann, OU=Android developer, O=Uni HH, L=Hamburg, ST=Hamburg, C=DE
Geben Sie das Passwort für <alias_name> ein.
(EINGABETASTE, wenn Passwort dasselbe wie für Keystore):
Geben Sie das Passwort erneut ein:
[my-release-key.keystore wird gesichert.]

Kommandozeilenfenster während der Ausführung von Keytool.exe

Nachdem der Keystore mit dem privaten Schlüssel erzeugt worden ist, kann jetzt die fertige Android Anwendung im release mode kompiliert werden. Fertige Android Anwendungen haben die Endung „.apk“.

Dazu klickt man in Eclipse in dem Package Explorer mit der rechten Maustaste auf Projekt welches exportiert werden soll, *Android Tools > Export Unsigned Application Package.*

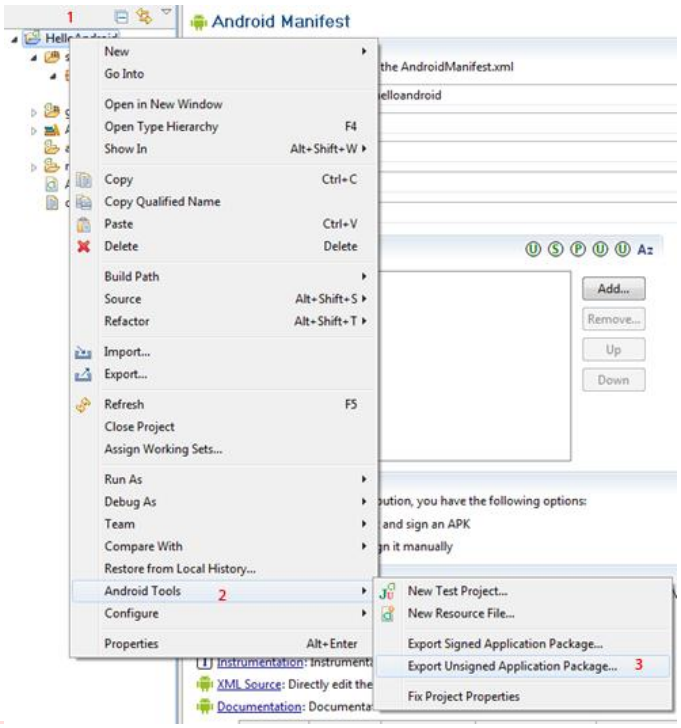


Abbildung 4. Exportieren einer unsignierten Anwendung.

Alternativ geht es im *Android Manifest* > *Exporting* > *Export an unsigned APK.*

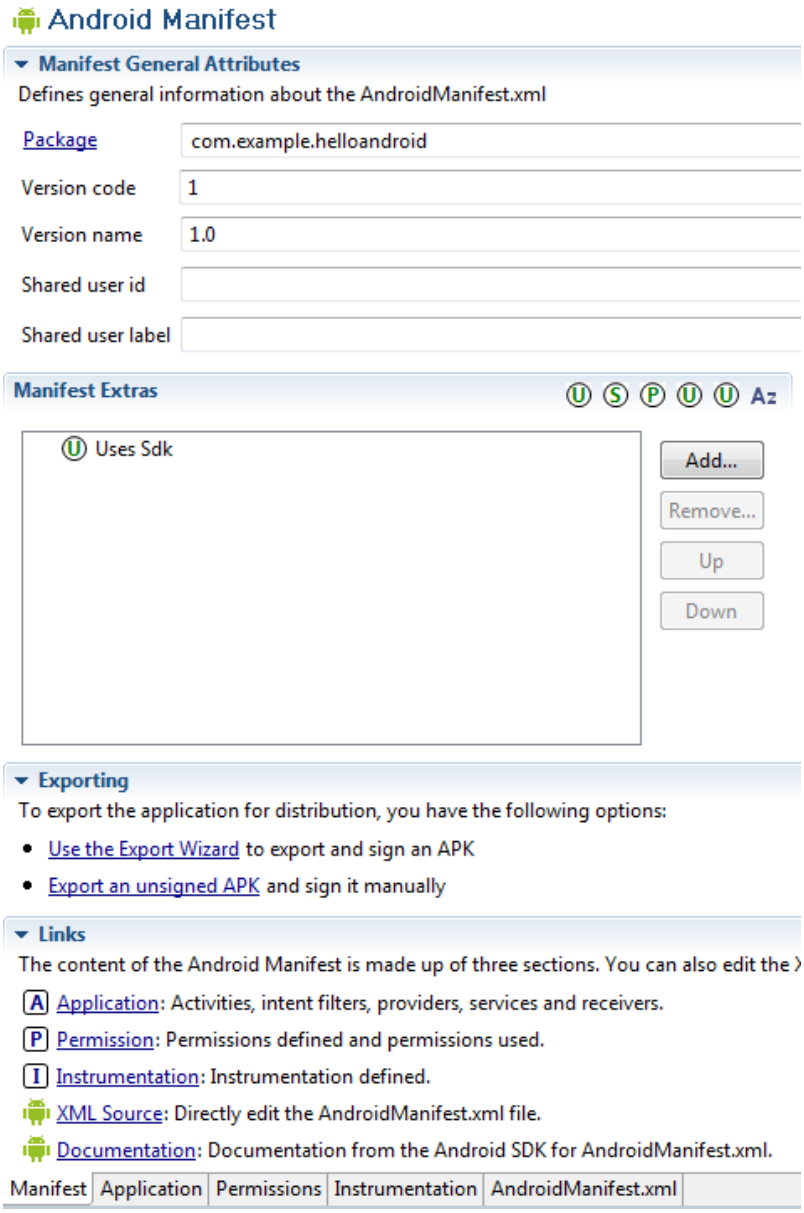


Abbildung 5. Exportieren einer unsignierten Anwendung über die Manifest-Ansicht

Apk Datei signieren

Nachdem die Anwendung nun als apk Datei vorliegt, kann diese signiert werden. Dafür wird ein weiteres Tool *jarsigner.exe* benötigt. Jarsigner liegt in JavaSDK/bin Ordner.

Mit diesem Befehl wird die Apk Datei signiert:

```
jarsigner.exe -verbose -keystore my-release-key.keystore my_application.apk
alias_name
```

Dabei muss man den zuvor angelegten Keystore, die zu signierende Anwendung und den gewünschten Schlüssel angeben. In unserem Fall ist das Keystore: my-release-key.keystore, Anwendung: my_application.apk und Schlüssel: alias_name.

Die Konsolenausgabe sieht so aus:

```
PS C:\Program Files\Java\jdk1.6.0_18\bin> jarsigner.exe -verbose -keystore
my-release-key.keystore C:\HelloAndroid.apk alias_name
Enter Passphrase for keystore:
Enter key password for alias_name:
  adding: META-INF/ALIAS_NA.SF
  adding: META-INF/ALIAS_NA.RSA
  signing: res/layout/main.xml
  signing: AndroidManifest.xml
  signing: resources.arsc
  signing: res/drawable-hdpi/icon.png
  signing: res/drawable-ldpi/icon.png
  signing: res/drawable-mdpi/icon.png
  signing: classes.dex
```

Nachdem die Anwendung signiert wurde, muss diese Anwendung noch optimiert werden. Dafür ist wird das Tool zipalign.exe benutzt. Zipalign.exe liegt in AndroidSDK/tools Ordner.

Mit folgendem Kommando, wird das Programm optimiert. Der erste Parameter (your_project_name-unaligned.apk) gibt die noch nicht optimierte Anwendung, der zweite Parameter (your_project_name.apk) die optimierte Anwendung an.

```
zipalign.exe -v 4 your_project_name-unaligned.apk   your_project_name.apk
```

Weitere Informationen zur Zipalign.exe findet man in dem Android Deleloper Blog[3].

3 http://android-developers.blogspot.com/2009/09/zipalign-easy-optimization.html

2 Publishing Voraussetzungen

Die zuvor gemachten Schritte wie das Anlegen eines Keystores inkl. Schlüsseln, Signieren und Optimieren der Anwendung, kann man bequem mit Eclipse ADT Plugin erledigen.

Dabei klickt man mit der rechten Maustaste auf das Projekt > Android Tools und wählt Export Signed Application Package aus. Alternativ geht es über: File > Export > Android > Export Android Application. Nach der Auswahl des zu exportierenden Projektes, wird man gefragt, ob ein vorhandener Keystore benutzt werden soll oder ein neuer angelegt werden soll.

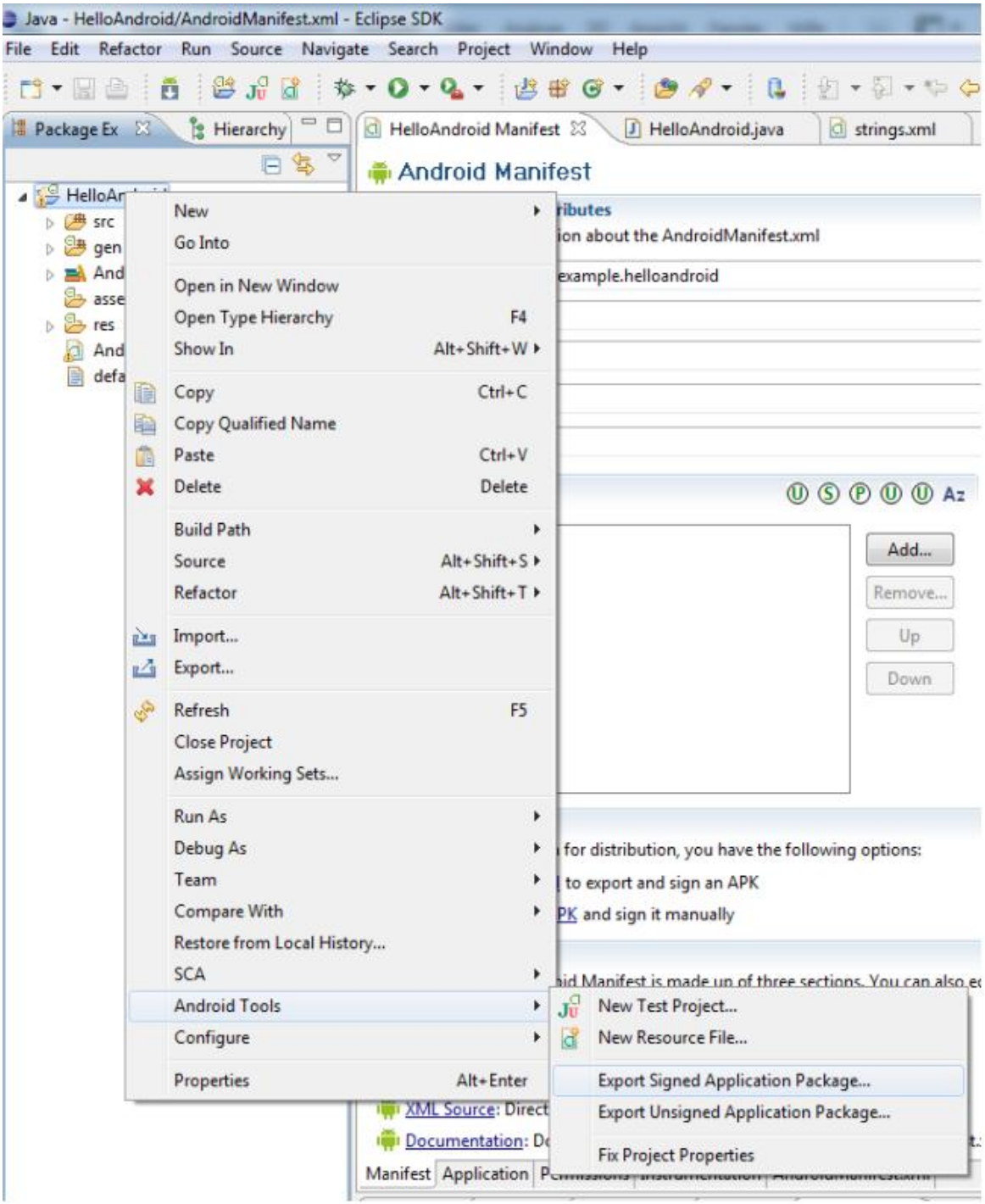


Abbildung 6. Exportieren einer Anwendung mit ADT

3 Publishing

3.1 Android Market

Um eigene Anwendung auf Android Market zu veröffentlichen, gibt es einige Voraussetzungen die zuerst erfüllt werden müssen. Hat man die obige Voraussetzungsliste abgearbeitet, so erübrigen sich bereits viele Punkte.

Voraussetzungen vom Android Market Server:

- Anwendung muss signiert sein und Zertifikat-Ablaufdatum nach 22.10.2033
- *android:versionCode* und *android:versionName* müssen definiert sein
- *android:icon* und *android:label* müssen definiert sein

Weitere Voraussetzungen:

- Google Konto
- kostenpflichtige Registrierung als Entwickler auf Android Market

3.2 Endgerät

Um eine Fertige Anwendung auf ein Endgerät zu übertragen, gibt es drei Möglichkeiten.

Android Debug Bridge Tool

Die erste Möglichkeit ist das Android Debug Bridge Tool (adb.exe) welches in Android SDK/tools Ordner liegt, zu benutzen. Damit lassen sie Anwendungen auf den Emulator oder angeschlossenes Gerät installieren. Folgende Befehle installieren die Anwendung „Myapp.apk“:

adb.exe –e install Myapp.apk installiert auf den Emulator *adb.exe –d install Myapp.apk* installiert auf ein verbundenes Physisches Gerät

Falls die Installation nicht klappen sollte, muss man auf dem Android Gerät unter *Einstellungen* > *Anwendungen* > die Option *Unbekannte Quellen* aktivieren.

Dalvik Debug Monitor Server

Die zweite Möglichkeit Anwendungen auf Endgeräte zu installieren, stell der Dalvik Debug Monitor Server (DDMS) dar.

Zunächst muss in Eclipse die DDMS Perspektive geöffnet werden. Danach unter Device das Gerät auswählen auf dem das Programm installiert werden soll. Dann im File Explorer nach Data/app Ordner navigieren und oben rechts den Knopf („*push a file onto the device*“) betätigen. Zum Schluss wird man aufgefordert die Anwendung auszuwählen, welche installiert werden soll.

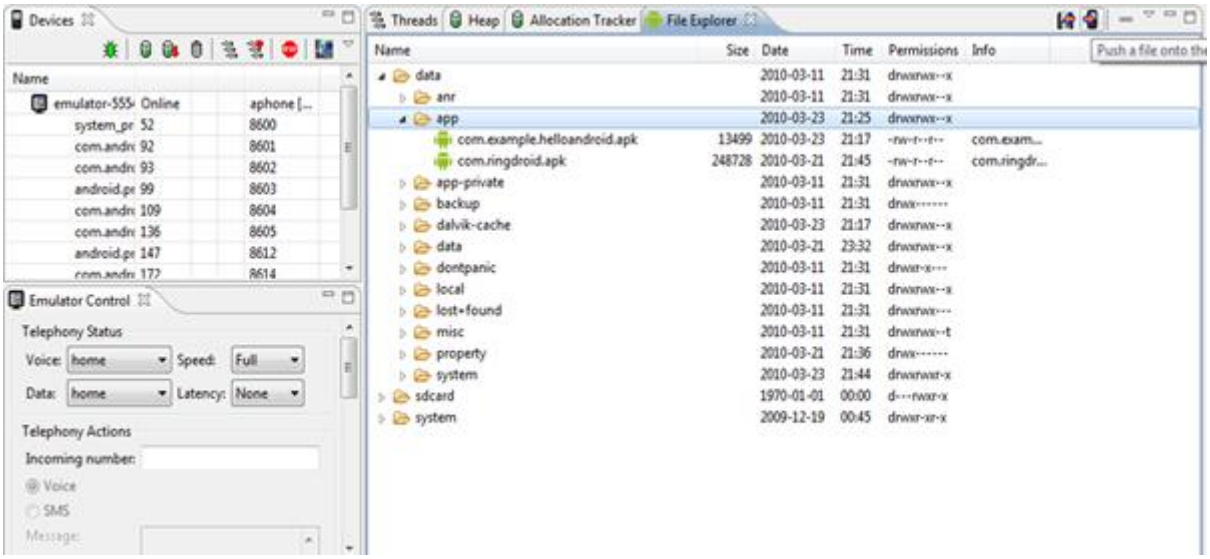


Abbildung 7. Eclipse in DDMS Perspektive

Weitere Informationen zur ADB, DDMS und vielen anderen Tools, findet man auf der Android Developer Homepage[4].

Von SD Karte installieren

Die letzte Möglichkeit wie man eine Anwendung installiert ist ebenfalls einfach. Es wird eine fertige und signierte apk Datei auf die Speicherkarte kopiert. Danach kann man mit verschiedenen Programmen wie z.B. AppInstaller, apk Dateien von SD Karte installieren.

4 http://developer.android.com/guide/developing/tools/ddms.html

4 Literaturverzeichnis

- http://www.android.com/
- http://www.devx.com/wireless/Article/39972/1954
- http://www.androiddevelopment.org/2009/01/19/signing-an-android-application-for-real-life-mobile-device-usage-installation/
- Arno Becker, Marcus Pant: „*Android Grundlagen und Programmierung*“ 1 Auflage 2009 dpunkt.verlag