

Automatisierte Abläufe mit Ant

U. Breymann H. Mosemann

Ergänzendes Material zum Buch »Java ME«
<http://www.java-me.de>

Ant ist ein Programm zur Automatisierung von Abläufen [Ant]. Es entspricht (sehr) grob dem *make*-Programm, das in der C-Welt gebräuchlich ist. Ant benötigt eine Datei, in der die Steuerungsinformationen für den Ablauf stehen. Ant nimmt an, dass diese Datei *build.xml* heißt – wenn nicht, muss Ant der Dateiname mit der Option `-f` übergeben werden, zum Beispiel `-f datei.xml`. Wie der Name andeutet, sind die Informationen im XML-Format hinterlegt. Hier soll an einem einfachen MIDlet, nämlich dem Beispiel aus dem Kapitel »Erste Schritte«, die Automatisierung der maschinellen Verarbeitung gezeigt werden. Ant ist ein praktisches und sehr mächtiges Werkzeug. Zum Einstieg und für die Arbeit mit Ant empfehlen wir das umfangreiche und dennoch gut lesbare Werk von Hatcher und Loughran [HaLo]. Eclipse enthält Ant und kann eine Ant-Build-Datei ausführen, so dass die Möglichkeiten der IDE dadurch erheblich erweitert werden.

Eine Build-Datei enthält eine Menge zu erreichender *Ziele*. Zu jedem Ziel (engl. *target*) wird angegeben, mit welchen Aktionen es erreicht wird, und außerdem, welche andere Ziele vorher erreicht sein müssen. Die dazu notwendigen Aktionen werden ggf. ausgeführt. Beispiel: Das Packen von class-Dateien mit `jar` (ein Ziel/Target) setzt voraus, dass erst die Java-Dateien erfolgreich kompiliert worden sind (ein anderes Ziel). Am Anfang werden einige Namen im Stil der Java-Properties vereinbart. Zum Beispiel kann der Name `ziel.verzeichnis` den Wert `dest` bekommen. Ab diesem Punkt in der Datei kann der Wert überall mit `${ziel.verzeichnis}` eingesetzt werden. Der Vorteil ist, dass nur die Definition des Namens verändert werden muss, sollte sich zum Beispiel der Name des Verzeichnisses ändern. Nun zu dem Beispiel der Build-Datei für das MIDlet-Projekt *Projekt1*, die Stück für Stück erläutert wird und die zusammen mit dem Beispiel auf der Internetseite zum Buch zu finden ist (<http://www.java-me.de/>):

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <project name="Projekt1" default="jad" basedir=".">
3   <!-- einige Namen deklarieren -->
4   <!-- wtkhome -->
5   <property file="${user.home}/build.properties"/>
6   <property file="build.properties"/>   <!-- wtk.midpapi -->
```

```

7 <property name="wtklib" value="${wtkhome}/lib"/>
8 <property name="projektname" value="Projekt1"/>
9 <property name="packagename"
10     value="de.java_me.jme.beispiele.Projekt1"/>
11 <property name="midlet" value="${packagename}.ErstesMIDlet"/>
12 <property name="uri" <!-- URI für OTA-Zugriff: -->
13     value="http://localhost/~uli/jme/${projektname}.jad"/>

```

Zeile 1 ist der Beginn jeder XML-Datei¹. Der Wert `iso-8859-1` des Attributs `encoding` gewährleistet, dass der XML-Interpreter nicht abstürzt oder abbricht, wenn die Datei Umlaute enthält. Zeile 2 definiert den frei wählbaren Projektnamen, das zu erreichende Ziel, falls kein anderes beim Aufruf angegeben wird, und das Arbeitsverzeichnis. Die Zeilen 3 und 4 sind XML-Kommentare und werden von Ant ignoriert. Anschließend werden zwei Property-Dateien eingelesen. In diesen stehen Properties und ihre Werte in einer einfachen Syntax, zum Beispiel:

```

wtkhome=/home/uli/wtk
wtklib=/home/uli/wtk/lib
wtk.midpapi=${wtklib}/cldcapi10.jar:${wtklib}/midpapi20.jar

```

Die folgenden Zeilen definieren weitere Namen, wobei in den Definitionen die vorhergehenden Namen mit Hilfe der Syntax `${Name}` benutzt werden.

```

14 <!-- Verzeichnisnamen und Pfade usw deklarieren -->
15 <property name="src" value="src"/>
16 <property name="build.dir" value="tmpclasses"/>
17 <property name="verified.dir" value="classes"/>
18 <property name="task.dir" value="anttasks"/>
19 <property name="jad.dir" value="jad"/>
20 <property name="deployed.dir" value="bin"/>
21 <property name="javadoc.dir" value="docs"/>
22 <property name="tmplib.dir" value="tmpplib"/>
23 <property name="resource.dir" value="res"/>
24 <property name="compile.classpath" value="${wtk.midpapi}"/>
25 <property name="execclasspath" value="${wtk.midpapi}"/>

```

Jetzt folgen die einzelnen Targets. Das Ziel `init` hängt von nichts ab. Es legt nur das Verzeichnis für die class-Dateien an. Das Attribut `description` dient zur Dokumentation und zur Hilfe: Das Kommando

```
ant -projecthelp
```

(oder kurz `ant -p`) listet die Beschreibungen aller Ziele auf.

```

26 <target name="init"
27     description="build-Verzeichnis erzeugen">

```

¹ Die Zeilennummern sind nicht Teil der Datei.

```

28 <mkdir dir="${build.dir}"/>
29 </target>

```

Das Ziel `compile` hängt von `init` ab, das heißt, wenn mit dem Befehl `ant compile` die Übersetzung angestoßen wird, wird automatisch vorher das Build-Verzeichnis mit `init` angelegt, falls es noch nicht existiert. Das Ziel zur Compilation definiert das Quellverzeichnis (und damit alle dazugehörigen Unterverzeichnisse) und weitere Parameter. `javac` ist eine fest eingebaute so genannte Ant-Task, die im Namen mit dem Aufruf des Java-Compilers übereinstimmt.

```

30 <target name="compile" depends="init"
31   description="alles compilieren">
32   <javac srcdir="${src}" destdir="${build.dir}" debug="on"
33     bootclasspath="${compile.classpath}"
34     source="1.3"
35     target="1.1">
36   </javac>
37 </target>

```

Boot-Classpath, Source- und Target-Version für Java sind unbedingt notwendig, damit das übersetzte Programm auf einem mobilen Gerät mit seinen Einschränkungen lauffähig ist. Weglassen dieser Parameter würde bedeuten, dass der Classpath und die Versionsnummer der aktuell benutzten Java SE verwendet würden. Die VM des Geräts unterstützt nicht, wie schon beschrieben, den Umfang der Java SE, sondern nur eine Teilmenge davon, und braucht überdies spezielle Bibliotheken. Ant muss alles mitgeteilt werden, was als Optionen auch für einen Kommandozeilenbefehl benötigt wird. Der Vorteile von Ant gegenüber einem Batch-Skript liegt unter anderem in der Definition der Abhängigkeiten. Im folgenden Ziel wird die Präverifikation durchgeführt. Dabei ist `exec` eine Ant-Task, die beliebige Programme aufrufen kann. Es können Argumente und Optionen übergeben werden.

```

38 <target name="verifizieren" depends="compile"
39   description="Prä-Verifikation durchführen">
40   <mkdir dir="${verified.dir}"/>
41   <exec executable="${wtkhome}/bin/preverify">
42     <arg value="-classpath"/>
43     <arg value="${execclasspath}"/>
44     <arg value="-d"/>
45     <arg value="${verified.dir}"/>
46     <arg value="${build.dir}"/>
47   </exec>
48 </target>

```

Auch das nächste Ziel, die Erzeugung einer Manifest-Datei, wird von einer fest eingebauten Ant-Task realisiert. Die Manifest-Datei wird für das ausführbare Java-Archiv, das im nächsten Schritt erzeugt wird, benötigt.

```

49 <target name="manifest" depends="verifizieren"
50     description="Manifest erzeugen">
51     <mkdir dir="{verified.dir}/META-INF"/>
52     <manifest file="{verified.dir}/META-INF/MANIFEST.MF">
53         <attribute name="Manifest-Version" value="1.0"/>
54         <attribute name="MicroEdition-Configuration"
55             value="CLDC-1.0"/>
56         <attribute name="MIDlet-Name"
57             value="{projektname} Midlet Suite"/>
58         <attribute name="MIDlet-Vendor" value="UB"/>
59         <attribute name="MIDlet-1" value="{projektname},
60             {projektname}.png, {midlet}"/>
61         <attribute name="MIDlet-Version" value="1.0.0"/>
62         <attribute name="MicroEdition-Profile" value="MIDP-2.0"/>
63     </manifest>
64 </target>

```

Das Packen aller class-Dateien und aller Ressourcen-Dateien übernimmt die Ant-Task `jar`. Eine Liste von `filesets` gibt an, welche Dateien einzuschließen sind:

```

65 <target name="packen" depends="manifest"
66     description="als jar-Datei packen">
67     <mkdir dir="{tmplib.dir}"/>
68     <jar destfile="{tmplib.dir}/{projektname}.jar"
69         manifest="{verified.dir}/META-INF/MANIFEST.MF">
70         <fileset dir="{verified.dir}">
71             <include name="**/*.class"/>
72         </fileset>
73         <fileset dir="{resource.dir}">
74             <include name="**/*"/>
75         </fileset>
76     </jar>
77 </target>

```

Es gibt eine Menge vordefinierter Ant-Tasks, aber das reicht manchmal nicht. Es muss eine `jad`-Datei erzeugt werden, die die Größe der `jar`-Datei nach dem obfuscate-Prozess enthält. Ant bietet die Möglichkeit, eigene Programme als Ant-Task zu schreiben, die als `jar`-Datei zur Ausführung gebracht werden. Das nächste Ziel erledigt das Übersetzen und Packen einer Ant-Task, die die `jad`-Datei erzeugt. Der Quellcode dieser selbstgeschriebenen Ant-Task und einige Erläuterungen folgen auf Seite 7.

```

78 <target name="maketask" depends="init"
79     description="ant-Tasks erzeugen">
80     <javac srcdir="{task.dir}" destdir="{task.dir}" debug="on">
81     </javac>

```

```

82 <jar destfile="${task.dir}/MakeJad.jar"
83     manifest="${task.dir}/MANIFEST.MF">
84     <fileset dir="${task.dir}">
85         <include name="**/*.class"/>
86     </fileset>
87 </jar>
88 </target>

```

Als nächstes folgt das Ziel `obfuscate` zum Ersetzen aller Klassennamen mit Ausnahme der Hauptklasse und Komprimierung. Die Ant-Task `java` startet den Proguard-Obfuscator. Das Attribut `fork` muss auf `true` gesetzt werden, damit zum Starten eine neue Instanz der VM verwendet wird.

```

89 <target name="obfuscate" depends="packen"
90     description="obfuscate mit proguard">
91     <mkdir dir="${deployed.dir}" />
92     <java jar="${wtkhome}/bin/proguard.jar"
93         failonerror="true"
94         fork="true">
95         <arg value="-libraryjars" />
96         <arg value="{execclasspath}" />
97         <arg value="-injars" />
98         <arg value="{tmplib.dir}/{projektname}.jar" />
99         <arg value="-outjar" />
100        <arg value="{deployed.dir}/{projektname}.jar" />
101        <arg value="-keep public class * extends
102            javax.microedition.midlet.MIDlet" />
103    </java>
104 </target>

```

Das Ziel `jad` erzeugt die `jad`-Datei sowie die `html`-Datei für den OTA-Zugriff mit Hilfe der selbstgeschriebenen Ant-Task. Nach der Taskdefinition wird die Task aufgerufen, wobei ihr hier drei Parameter mitgegeben werden, die sie zur Erfüllung der Aufgabe benötigt.

```

105 <target name="jad" depends="obfuscate,maketask"
106     description="jad- und html-Datei erzeugen">
107     <taskdef name="makejad" classname="MakeJad"
108         classpath="{task.dir}/MakeJad.jar" />
109     <makejad projektname="{projektname}"
110         manifestdir="{verified.dir}/META-INF"
111         destdir="{deployed.dir}" />
112 </target>

```

Das Target `install` kopiert alle für einen OTA-Zugriff benötigten Dateien in das Verzeichnis `public_html/jme` des Benutzers und macht diese Dateien damit öffentlich zugreifbar. Das Verzeichnis muss dem Web-Server bekannt sein.

```

113 <target name="install" depends="jad"
114     description="kopiere jad, jar und html nach public_html/jme">
115     <copy todir="${user.home}/public_html/jme">
116         <fileset dir="${deployed.dir}"/>
117     </copy>
118 </target>

```

Das Target `run` startet mit Hilfe der schon oben verwendeten Ant-Task `exec` den Emulator. Das Target `runOTA` startet den Emulator, indem der OTA (Over the Air)-Mechanismus simuliert wird. Der Web-Server muss dazu gestartet sein, auch müssen die MIME-Typen wie beschrieben konfiguriert worden sein. Das MIDlet darf noch nicht installiert worden sein.

```

119 <target name="run" depends="jad"
120     description="MIDlet im Emulator laufen lassen">
121     <exec executable="${wtkhome}/bin/emulator">
122         <arg value="-Xdescriptor"/>
123         <arg value="${deployed.dir}/${projektname}.jad"/>
124         <arg value="-classpath"/>
125         <arg value="${deployed.dir}/${projektname}.jar"/>
126     </exec>
127 </target>
128
129 <target name="runOTA" depends="jad"
130     description="MIDlet von localhost in den Emulator laden
131     und laufen lassen">
132     <exec executable="${wtkhome}/bin/emulator">
133         <arg value="-Xjam:transient=${uri}"/>
134     </exec>
135 </target>

```

Der Vollständigkeit halber folgt das Target `javadoc` zum automatisierten Erstellen der Dokumentation.

```

136 <target name="javadoc" depends="compile"
137     description="Java-Dokumentation erzeugen">
138     <mkdir dir="${javadoc.dir}"/>
139     <javadoc author="true"
140         destdir="${javadoc.dir}"
141         private="false"
142         packagenames="${packagename}.*"
143         sourcepath="src"
144         classpath="${compile.classpath}">
145     </javadoc>
146 </target>

```

Das letzte Ziel `clean` ist zum Aufräumen da – alle temporären Verzeichnisse werden gelöscht. Nicht existierende Verzeichnisse werden ignoriert.

```
147 <target name="clean"
148     description="temporäre Verzeichnisse löschen">
149     <delete dir="${build.dir}"/>
150     <delete dir="${verified.dir}"/>
151     <delete dir="${javadoc.dir}"/>
152     <delete dir="${tmp.lib.dir}"/>
153     <delete dir="${deployed.dir}"/>
154 </target>
155
156 </project> <!-- Ende des Ant-Skripts -->
```

0.0.1 Ant-Task selbst schreiben

Die Ant-Task `MakeJad` erzeugt die `jad`-Datei aus der Manifest-Datei. Es werden nur der MIDlet-jar-Dateiname und die Größe der jar-Datei hinzugefügt. Eine Ant-Task genügt folgenden Bedingungen:

- Sie erbt von `org.apache.tools.ant.Task`.
- Es muss für jeden übergebenen Parameter (siehe Aufruf oben) eine entsprechende `set... (String)`-Methode geben.
- `org.apache.tools.ant.BuildException` hilft Ant, bei Fehlern mit einer aussagekräftigen Meldung abzubrechen.
- Es muss eine Methode `public void execute()` geben. Dies ist die Methode, die im Build-Prozess aufgerufen wird.

```
import org.apache.tools.ant.Task;
import org.apache.tools.ant.BuildException;
import java.io.*;

public class MakeJad extends Task {
    private String manifestdir= null;
    private String destDir= null;
    private String projektname = null;

    public void setManifestdir(String d) {
        manifestdir = d;
    }

    public void setDestDir(String z) {
        destDir = z;
    }
}
```

```

public void setProjektname(String z) {
    projektname = z;
}

public void erzeugeHtmlDatei() {
    String ausgabe = destDir + "/" + projektname + ".html";
    PrintStream ps = null;
    try {
        ps = new PrintStream(new FileOutputStream(ausgabe));
        ps.print("<html><head><title>");
        ps.print(projektname);
        ps.print("</head><body>\n<a href=\"");
        ps.print(projektname);
        ps.print(".jad\">");
        ps.print(projektname);
        ps.println(".jad</a>\n</body></html>");
        System.out.println(ausgabe + " erzeugt");
    } catch(IOException e) {
        throw new BuildException("Task makejad: "
            + "erzeugeHtmlDatei(): IOException!");
    } finally {
        if(ps != null)
            ps.close();
    }
}

public void erzeugeJadDatei() {
    // zuerst Manifest-Datei kopieren
    StringBuffer sb = new StringBuffer();
    String quelle = manifestdir + "/MANIFEST.MF";
    FileInputStream fi = null;
    try {
        fi = new FileInputStream(quelle);
    } catch(IOException e) {
        System.out.println("Datei " + quelle
            + " kann nicht geöffnet werden!");
        System.exit(1);
    }
    PrintStream ps = null;
    try {
        // Manifest-Datei lesen
        int einByte = fi.read();
        while(einByte != -1) {
            sb.append((char)einByte);
            einByte = fi.read();
        }
        fi.close();
    }
}

```

```

String temp = sb.toString();
/* Berücksichtigen, dass entsprechend der Definition
   von Manifest-Dateien eine Zeile mit mehr als 72
   Zeichen in der nächsten Zeile, die dann mit einem
   Leerzeichen beginnt, fortgesetzt wird. D.h.,
   solcher Art Zeilenumbrüche beseitigen: */
String zeilen = temp.replaceAll("\r\n ", "");
String ausgabe = destDir + "/" + projektname + ".jad";

ps = new PrintStream(new FileOutputStream(ausgabe));
ps.print(zeilen);
ps.print("MIDlet-Jar-URL: ");
String jarfilename = projektname + ".jar";
ps.println(jarfilename);
File jarfile = new File(destDir + "/" + jarfilename);
if(!jarfile.exists()) {
    throw new BuildException("Task makejad: "
        + jarfilename + " existiert nicht");
}
ps.print("MIDlet-Jar-Size: ");
ps.println(jarfile.length());
System.out.println(ausgabe + " erzeugt");
} catch(IOException e) {
    throw new BuildException("Task makejad: "
        + "erzeugeJadDatei(): IOException!");
} finally {
    if(ps != null)
        ps.close();
}
}

public void execute() {
    if(destDir == null) {
        throw new BuildException("Task makejad: Attribut "
            + "destDir nicht definiert!");
    }
    if(manifestdir == null) {
        throw new BuildException("Task makejad: Attribut "
            + "manifestdir nicht definiert!");
    }
    if(projektname == null) {
        throw new BuildException("Task makejad: Attribut "
            + "projektname nicht definiert!");
    }
}

```

```
        erzeugeJadDatei ();  
        erzeugeHtmlDatei ();  
    }  
}
```

Literatur

[Ant] Ant Homepage, *<http://ant.apache.org>*

[HaLo] Erik Hatcher, Steve Loughran: *Java Development with Ant*. Manning 2003