

2 Der erste Kontakt

Umgangssprachlich ausgedrückt ist es die Aufgabe von Ant, Gruppen von Dateien zu spezifizieren und auf diese Dateien bestimmte Kommandos anzuwenden. Zu diesen Kommandos gehören der Aufruf diverser Java-Compiler, Kopierkommandos sowie die Erstellung von Zip- und anderen Archiven. Ant ist daher hervorragend geeignet, um aus Java-Quellcode eine vollständige, verteil- und installierbare Anwendung zu erstellen und sogar die Verteilung (Deployment) bzw. Installation selbst zu übernehmen. Darüber hinaus sind auch Einsatzgebiete denkbar, die nichts mit der Programmierung im engeren Sinne zu tun haben. So können mit Ant sehr effektiv Dateien kopiert werden, Ersetzungen in Property-Dateien stattfinden etc.

2.1 Installation

Das Build-Tool Ant ist eine reine Java-Anwendung. Es existiert kein explizites Installations-Tool. Vor der ersten Benutzung sind daher einige Vorbereitungen zu treffen.

Um alle Möglichkeiten von Ant nutzen zu können, muss auf Ihrem Rechner ein JDK installiert sein. Alle Ant-Versionen ab 1.6 erfordern zwingend eine Java-Version ab 1.2.

Besonders komfortabel kann Ant eingesetzt werden, wenn ein JDK ab der Version 1.4 benutzt wird. In dieser Version sind einige Pakete enthalten, die bei älteren Versionen separat installiert werden müssen, z.B. Pakete zur Auswertung regulärer Ausdrücke.

Üblicherweise ist Ant als Zip- oder Tar-Archiv verfügbar. Die aktuellste Version finden Sie auf der Homepage des Ant-Projektes (<http://ant.apache.org>). Welches konkrete Archiv Sie benutzen, hängt im Wesentlichen vom Betriebssystem und den verfügbaren Entpackern ab.

Das jeweilige Archiv entpacken Sie in das gewünschte Zielverzeichnis. Innerhalb des Archivs liegen alle Dateien unterhalb eines Wurzelverzeichnisses, dessen

Name aus der Zeichenkette *apache-ant-* und der Versionsnummer des Tools besteht. Die eindeutige Namensgebung des Installationsverzeichnisses ermöglicht es Ihnen, die neue Ant-Version parallel zu einer älteren Version zu installieren. Die Werte zweier Umgebungsvariablen (*PATH* und *ANT_HOME*) legen fest, welche Version benutzt wird.

Zum Start von Ant muss neben den Java-spezifischen Einstellungen die Umgebungsvariable *ANT_HOME* gesetzt werden. Ob das in den grundlegenden Systeminstellungen geschieht oder ob Sie beim Start des Befehlsinterpreters entsprechende Werte angeben, bleibt Ihnen überlassen. Auf jeden Fall müssen Sie die Umgebungsvariable *ANT_HOME* auf den Wurzelpfad des Ant-Paketes setzen. Das ist, sofern es nicht umbenannt wurde, das *apache-ant-x.x.x*-Verzeichnis. Außerdem muss das unter dem Ant-Wurzelverzeichnis liegende Unterverzeichnis *bin* in den Systempfad aufgenommen werden, damit das Ant-Startskript vom System gefunden wird.

Sind die Anpassungen erfolgt, können Sie einen einfachen Funktionstest durchführen. Rufen Sie von einem beliebigen Verzeichnis außerhalb des *bin*-Verzeichnisses von Ant einfach das Kommando

```
ant
```

auf. Sofern Ant vom System gestartet werden kann, sollte folgende Fehlermeldung erscheinen:

```
Buildfile: build.xml does not exist!  
Build failed
```

Falls diese Ausgabe erscheint, ist Ant korrekt installiert. Sollte sich im aktuellen Verzeichnis allerdings eine Datei mit dem Namen *build.xml* befinden, so versucht Ant, diese Datei auszuführen. Dies bewirkt allerdings ebenfalls Konsolenausgaben, die auf den korrekten Start von Ant hinweisen.

2.2 Grundlegende Elemente von Ant

Ant verwendet eine Reihe spezieller Elemente, die trotz eines allgemein gültigen Namens eine für Ant spezifische Bedeutung haben. Vor der Beschreibung der konkreten Funktionen ist es daher notwendig, zunächst die grundlegenden Begriffe zu erklären.

2.2.1 Projekt

Das *Projekt* ist das umfassende Objekt, dem alle anderen Elemente untergeordnet sind. Es definiert eine Ablaufumgebung für die Kommandos, die während des Builds ausgeführt werden.

Im Normalfall wird ein Projekt in genau einer Build-Datei abgelegt. Es ist möglich, aus einem Projekt heraus Elemente anderer Projekte zu nutzen, allerdings wird dabei im Standardfall eine separate Umgebung erzeugt.

2.2.2 Target

Ein *Target* ist ein Anweisungsblock, der ein oder mehrere Kommandos enthält. Die Targets selbst sind nur Hüllen oder Container; die eigentliche Arbeit wird stets von den Kommandos im Target ausgeführt, den so genannten Tasks.

Targets werden durch einen Namen identifiziert, der innerhalb der Build-Datei eindeutig sein muss. Da ein Target immer nur im Zusammenhang mit einer Build-Datei verwendet werden kann, dürfen sich die Namen der Targets in verschiedenen Dateien wiederholen, ohne dass es zu Mehrdeutigkeiten kommt.

Ant kann Targets in Abhängigkeit von bestimmten Rahmenbedingungen und Zuständen ausführen und damit eine einfache Ablaufsteuerung realisieren.

2.2.3 Task

Ein *Task* ist ein ausführbares Kommando. Ant verfügt über eine Vielzahl von vorgefertigten Tasks für unterschiedliche Aufgaben. Durch Attribute und Sub-Tags werden die Tasks mit Parametern versorgt.

Tasks werden normalerweise nur innerhalb von Targets benutzt. Nur so ist die geordnete Ausführung der Kommandos möglich. Es ist allerdings möglich, Tasks auch außerhalb von Targets zu platzieren. Diese Tasks werden immer ausgeführt, wenn eine Build-Datei gestartet wird.

2.2.4 Datatype

Der Ant-Begriff *Datatype* ist nicht direkt mit den Datentypen vergleichbar, die in herkömmlichen Programmiersprachen wie Java benutzt werden. In Ant ist ein Datentyp ein relativ komplexes Objekt, das Daten enthält. Der Datentyp beschafft die Daten selbst, er besitzt somit eine eigene Funktionalität und ist daher nicht passiv. Allerdings führt der Datentyp keine Aktionen auf diesen Daten aus. Seine Aufgabe ist es, die Daten einem umgebenden Task zur Verfügung zu stellen.

Ein Beispiel für einen Ant-Datentyp ist das *Fileset*. Ein Fileset selektiert Dateien, wobei relativ komplexe Selektionsmechanismen zur Verfügung stehen.

Ant-Datentypen können außerhalb von Tasks und Targets notiert werden. Sie führen die Datenbeschaffung immer dann durch, wenn sie durch die Laufzeitumgebung abgearbeitet werden.

2.2.5 Property

Ein *Property* ist ein Platzhalter für einen Wert. Der Wert des Property kann durch verschiedene Mechanismen gesetzt werden. Nachdem einem Property ein Wert zugewiesen wurde, kann er nicht mehr verändert werden. Insofern ähnelt ein Property den aus herkömmlichen Programmiersprachen bekannten Konstanten.

2.2.6 Referenzen

Viele Ant-Elemente, insbesondere die Ant-Datentypen, können mit einer ID versehen werden. Über diese ID, in Ant auch als *Referenz* bezeichnet, ist der Zugriff auf das konkrete Element an beliebiger Stelle im Skript möglich. Auf diese Weise können Sie einen Datentyp (z.B. eine komplexe Definition für den Classpath) mehrfach verwenden. Außerdem erleichtert die Trennung von Definition und Verwendung die Wartung.

2.2.7 Kommandozeile

Ant ist ein Konsolenprogramm. Es wird über die Kommandozeile gestartet. Eine Reihe optionaler Parameter ermöglicht den gezielten Aufruf einzelner Targets oder das Erzeugen von Property.

2.3 Ein praktisches Beispiel

Das folgende kleine Beispiel soll zunächst eine einfache Build-Datei vorstellen, um Ihnen ein Gefühl für die Arbeitsweise von Ant zu geben. Der Sinn dieses Beispiels ist es nicht, die Kommandos im Detail zu erläutern. Dies bleibt den folgenden Kapiteln vorbehalten. Das Beispiel wurde bewusst einfach gehalten, ist aber funktionsfähig. Eine Build-Datei, die in einer realen Umgebung eingesetzt wird, wäre sicherlich etwas umfangreicher. Das Skript wurde, um die einzelnen Teile besser beschreiben zu können, in mehrere Abschnitte zerlegt.

Die durch das Beispiel zu lösende Aufgabe besteht darin, eine Java-Anwendung zu kompilieren, aus den Class-Dateien ein Jar-Archiv zu erstellen und anschließend die Anwendung zu testen. Der Build soll drei Unterverzeichnisse benutzen, jeweils eines für die Quell- und die Class-Dateien sowie eines für das Archiv. Alle befinden sich unter einer gemeinsamen Wurzel, unter der auch die Build-Datei liegt.

Eine Build-Datei wird zunächst durch eine XML-typische Anweisung eingeleitet, die vor allem die Zeichenkodierung festlegt. Diese Zeile muss in allen Build-Dateien enthalten sein. Sie wird in den weiteren Beispielen dieses Buches nicht mehr abgedruckt.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="bsp0201" default="main" basedir=".">
```

Der eigentliche Inhalt wird durch das Tag `<project>` eingeleitet. Das Attribut `default` definiert das zu startende Target (vergleichbar mit einem Unterprogramm). Das Attribut `name` dient nur zur Information und hat keine praktische Bedeutung. Wichtig ist allerdings das Attribut `basedir`. Es legt das Arbeitsverzeichnis für das Skript fest. Normalerweise dient das Verzeichnis, in dem sich das Skript befindet, als Arbeitsverzeichnis. Diese Vorgabe können Sie durch eine Zuweisung zum `basedir`-Attribut überschreiben.

Am Beginn einer Build-Datei befinden sich oft Definitionen. In unserem Beispiel werden drei Propertyts angelegt, die am ehesten mit Konstantendeklarationen einer herkömmlichen Programmiersprache vergleichbar sind und hier als Bezeichner für Verzeichnisnamen dienen. Propertyts können auch aus Dateien gelesen oder beim Aufruf der Build-Datei auf der Kommandozeile definiert werden. Somit ist es möglich, sie unabhängig von der eigentlichen Anwendung zu pflegen. Die Propertyts enthalten meist einfach strukturierte Werte.

```
<property name="dir.src" value="./source"/>
<property name="dir.build" value="./classes"/>
<property name="dir.lib" value="./lib"/>
```

Das nächste Element definiert eine komplexe Pfadliste. Konkret wird hier der Klassenpfad definiert. Für sich allein bewirkt dieses Element noch keine Aktion; es wird später von anderen Kommandos eingebunden. Dieses Verfahren erleichtert die spätere Anpassung und erspart Schreibarbeit.

```
<path id = "cp">
  <pathelement path = "${classpath}" />
  <pathelement location = "${dir.build}" />
</path>
```

In eine Pfadliste können Sie mittels diverser Sub-Tags Elemente aufnehmen, die beispielsweise auch die Selektion von Verzeichnissen über Suchmuster ermöglichen. In diesem Beispiel werden allerdings nur vorgegebene Pfade benutzt, die über Propertyts bereitgestellt werden. Der Bezug auf ein Property erfolgt, indem der Name in geschweifte Klammern eingeschlossen und ein `$`-Zeichen vorangestellt wird. Eines der beiden Propertyts (`dir.build`) wurde eingangs definiert, das andere (`classpath`) wird von Ant bereitgestellt. Es enthält den Classpath, der aus der gleichnamigen Systemvariablen entnommen wird.

Damit die Pfad-Definition später referenziert werden kann, muss per Attribut `id` eine eindeutige ID festgelegt werden. Im Beispiel wird hierfür die Bezeichnung `cp` verwendet.

Nun erst wird das erste Target angelegt. Es soll das Haupt-Target sein und erhält den Namen `main`. Somit wird es beim Start der Build-Datei aufgerufen, da dort als Default-Target ebenfalls `main` vorgegeben wurde.

```
<target name = "main"  
    depends = "prepare, compile, run" />
```

Dieses erste Target enthält keine weiteren Kommandos. Es dient hier nur dazu, die abhängigen Targets, die im Attribut `depends` angegeben wurden, auszuführen. Die drei Targets `prepare`, `compile` und `run` werden in diesem Beispiel in exakt dieser Reihenfolge ausgeführt. Die Ausführungsreihenfolge der Targets hängt allerdings von einigen Randbedingungen ab und kann in komplexen Anwendungen von der Notationsreihenfolge abweichen.

Das folgende Target `prepare` ist das erste, das wirklich Kommandos enthält. Kommandos werden bei Ant auch `Task` genannt. Das `prepare`-Target soll in diesem Beispiel demonstrieren, dass vor dem eigentlichen Kompilieren vorbereitende Arbeiten notwendig sein können. Das muss nicht nur das Löschen von Zielverzeichnissen sein. Denkbar ist auch das Lesen der Quelldateien aus einem Sourcecode-Verwaltungssystem, das Versionieren der alten Quelldateien o.Ä.

```
<target name = "prepare">  
    <mkdir dir="${dir.build}"/>  
    <delete>  
        <fileset  
            dir = "${dir.build}"  
            includes = "**/*"  
        />  
    </delete>  
</target>
```

In diesem Beispiel wird zunächst ein Verzeichnis angelegt, das später als Zielverzeichnis für den Compiler dient. Anschließend wird der Inhalt dieses Verzeichnisses, nicht aber das Verzeichnis selbst gelöscht. Die Kombination beider Kommandos stellt sicher, dass sowohl bei erstmaliger Ausführung als auch bei wiederholten Aufrufen immer ein leeres Build-Verzeichnis zur Verfügung steht. Das Löschen von alten Klassen ist empfehlenswert, da das Vorhandensein alter Class-Dateien mitunter zu unerwünschten Nebeneffekten führen kann. Zwingend notwendig ist es allerdings nicht.

Welche Dateien und Verzeichnisse gelöscht werden, wird hier durch ein so genanntes Fileset bestimmt. Ein Fileset enthält oft weitere Sub-Tags, mit denen Sie Dateien und Verzeichnisse auswählen können, und kann als Sub-Tag in vielen Kommandos enthalten sein. Es ist das vielleicht wichtigste und am meisten verwendete Tag überhaupt. In diesem Beispiel wird allerdings nur die einfachste Variante eines Filesets benutzt.

Nach dem `prepare`-Target wird das `compile`-Target aufgerufen. Es enthält drei Kommandos: den Aufruf des Java-Compilers, das Erzeugen eines weiteren Verzeichnisses und einen Befehl zum Erstellen eines Archivs.

```
<target name = "compile">
  <javac classpathref = "cp"
        destdir      = "${dir.build}"
        srcdir       = "${dir.src}"
        includes     = "**/*.java"
  />
  <mkdir dir="${dir.lib}"/>
  <jar destfile = "${dir.lib}/ae.jar">
    <fileset dir      = "${dir.build}"
            includes = "**/*.class"
  />
</jar>
</target>
```

Dem Java-Compiler müssen einige Angaben übermittelt werden. Hier sind es die Namen des Quell- und des Zielverzeichnisses, der Classpath und die Liste der zu kompilierenden Dateien. Für den Classpath wird das eingangs erzeugte Pfadelement benutzt. Der Zugriff erfolgt über die ID, die im Attribut `classpathref` eingetragen wird. Quell- und Zielverzeichnis werden per Property vorgegeben.

Nach dem Kompilieren wird ein Jar-Archiv erstellt. Es soll den Namen `ae.jar` erhalten und wird in einem separaten Verzeichnis abgelegt. Mittels eines Filesets werden alle Class-Dateien aus dem Build-Verzeichnis aufgenommen.

Sollte der Compiler einen Fehler erzeugen, bricht der Build sofort ab, die anderen Targets werden nicht ausgeführt, auch die Jar-Datei wird nicht erstellt.

Nach dem Kompilieren und Erstellen des Jar-Archivs kann die Anwendung getestet werden. Damit dieser Test nun nicht jedes Mal ausgeführt wird, arbeitet Ant dieses Target nur ab, wenn ein bestimmtes Property existiert. Erreicht wird dies durch das Attribut `if`. Es besagt, dass Ant das Target `run` nur dann ausführen soll, wenn das Property `test` existiert. Der konkrete Inhalt des Property ist dabei nicht von Bedeutung. Sie können dieses Property beim Aufruf der Anwendung in der Kommandozeile definieren und auf diese Weise flexibel entscheiden, ob im Anschluss an die Kompilierung ein Test stattfinden soll oder nicht.

```
<target name = "run"
        if = "test">
  <java classname = "AntExample"
        classpath = "${dir.lib}/ae.jar">
    <arg line = "${test}"/>
  </java>
</target>
```

Der Aufruf der Anwendung mit dem Java-Kommando ist relativ einfach. Natürlich muss der Name der zu startenden Klasse angegeben werden. Als Classpath reicht in diesem speziellen Fall die Jar-Datei aus. Da die zu testende Klasse Parameter auf der Kommandozeile erwartet, werden diese mit einem speziell dafür

vorgesehenen Sub-Tag `<arg>` übergeben. Es benutzt der Einfachheit halber den Inhalt des Property `test` als Übergabeparameter.

Der Test einer Anwendung durch einen einfachen Aufruf wird in der Praxis nicht ausreichen. Aber auf ähnliche Weise können auch komplexe Testwerkzeuge gestartet werden.

Nach diesem letzten Tag des Beispiels muss die XML-Datei jetzt nur noch korrekt abgeschlossen werden:

```
</project>
```

Im Ant-Paket sind Startdateien für die gängigsten Betriebssysteme enthalten. Sie können das Skript daher einfach mit

```
ant -f bsp0201.xml
```

starten. Damit wird das Target `run` allerdings nicht ausgeführt, da das Property `test` nicht existiert. Sie können das Property durch einen Parameter in der Kommandozeile erzeugen

```
ant -f bsp0201.xml -Dtest=Hallo
```

Bei dieser Form des Aufrufs wird allerdings der Compiler erneut aufgerufen. Um dies zu verhindern, können Sie gezielt ein anderes als das `main`-Target aufrufen. Dazu müssen Sie nur den Namen des auszuführenden Targets, z.B. `run`, nach dem Namen der Build-Datei angeben.

```
ant -f bsp0201.xml run -Dtest=Echo
```

Durch diesen Aufruf wird die `depends`-Kette des `main`-Targets unterdrückt. Ant führt nur das angegebene Target und dessen Sub-Targets aus, sofern welche existieren. In diesem Fall würde also nur der Aufruf der fertigen Anwendung erfolgen, eine Kompilierung findet nicht statt.

Nach diesem kurzen Einstieg finden Sie in den folgenden Kapiteln ausführliche Erläuterungen zu den einzelnen Kommandogruppen.