

## IN DIESER KURSEINHEIT

- *Einleitung*
  - Um was geht's eigentlich?
- *Installation*
  - Herunterladen
  - Entzippen
  - Testen
- *Erstes Beispiel*
  - Hello World Ausgabe
- *Zweites Beispiel*
  - Java HelloWorld
- *Was steckt alles in einem konkreten Build?*
  - ant -projecthelp
- *Starten mehrerer JVM's*
  - Client Server Beispiel
- *Tomcat als Ant Task*
  - Servlet Beispiel
- *Wie geht's weiter?*

## Apache Ant Starthilfe

### 1.1. Einleitung

Java basierte Software wird zunehmend mit einem Ant Skript (`build.xml`) ausgeliefert. Was ist Ant, was kann Ant, wie kann ich Ant sinnvoll einsetzen?

### 1.2. Installation

Der Anfang ist einfach: Ant ist gratis und kann von Apache heruntergeladen werden. Die Installation ist denkbar einfach: entzippen in ein Verzeichnis und in den Pfad eintragen!

- 1) <http://ant.apache.org/>
- 2) Binary Download von Ihrem Mirror Site :  
`www/mirror/apache/dist/ant/binaries/apache-ant-1.5.4-bin.zip`
- 3) Entzippen: bei mir ins Verzeichnis `c:\Ant`
- 4) Eintragen in den Pfad, damit Sie Ant überall nutzen können: `;%C:\Ant\bin;`
- 5) Die Umgebungsvariable `ANT_HOME` muss definiert werden: `ANT_HOME=C:\Ant`

Ich gehe davon aus, dass Java bereits installiert und `JAVA_HOME` definiert wurde. Ich habe `J2SDK 1.4.2` und `JAVA_HOME= C:\j2sdk1.4.2`

Testen Sie Ihre Installation: öffnen Sie ein DOS Fenster und tippen Sie ant ein:

- 1) `c:\ant` : Als Antwort erhalten Sie vermutlich  
`Buildfile: build.xml does not exist!`  
`Build failed`
- 2) Fragen Sie als Test die Version ab (die Ausgabe ist vermutlich nicht genau die selbe):  
`C:\>ant -version`  
`Apache Ant version 1.5.4 compiled on August 12 2003`

# ANT – DAS NEUE MAKE FÜR JAVA UND .NET

## 1.3. Ein erstes Beispiel – HelloWorld Ausgabe

Das einfachste Beispiel ist wie immer "Hello World". Wir wollen ant einsetzen, um die bekannte erste Testausgabe zu produzieren (alle Beispieldateien finden Sie im ZIP):

### Beispiel:

```
<!-- HelloWorld als ant Build.xml //-->

<!-- build.xml ist zwar ein XML Dokument aber ohne XML Header,
      der Header ist aber erlaubt)
<?xml version="1.0"?>  //-->

<!-- jedes Ant Skript besteht aus einem Projekt
      Pro Projekt muss ein Default Target definiert sein!
//-->
<project default="helloWorld">
<!-- jedes Ant Project besteht aus einer oder mehreren Targets,
      in denen Tasks ausgeführt werden
      Ant definiert viele Tasks : sie koennen diese einfach einsetzen
      Welche Tasks existieren, sehen Sie im Manual. Wir gehen auf einige ein
//-->
  <target name="helloWorld">
    <echo>
      Hello World
    </echo>
    <echo message="und so weiter..." />
  </target>
</project>
```

### Aufruf:

```
...HelloWorld>ant
Buildfile: build.xml
helloWorld:
    [echo]
    [echo]                Hello World
    [echo]
    [echo] und so weiter...
```

```
BUILD SUCCESSFUL
Total time: 1 second
```

Mit etwas mehr Ausgaben und mithilfe des Ant "Skripts" HelloWorld.xml:

```
..\HelloWorld>ant -v -buildfile HelloWorld.xml
Apache Ant version 1.5.4 compiled on August 12 2003
Buildfile: HelloWorld.xml
Detected Java version: 1.4 in: C:\j2sdk1.4.2\jre
Detected OS: Windows 2000
parsing buildfile HelloWorld.xml with URI =
file:C:/.../Ant/Beispiele/HelloWorld/HelloWorld.xml
Project base dir set to: C:\... \Ant\Beispiele\HelloWorld
Build sequence for target `helloWorld' is [helloWorld]
Complete build sequence is [helloWorld]
helloWorld:
    [echo]
    [echo]                Hello World
    [echo]
    [echo] und so weiter...
BUILD SUCCESSFUL
Total time: 1 second
```

# ANT – DAS NEUE MAKE FÜR JAVA UND .NET

Daraus erkennen wir auch gleich eine weitere Angabe, die typischerweise in einem Ant Skript gemacht wird: die Angabe des `base dir` (auf dieses Verzeichnis beziehen sich in der Regel weitere Verzeichnisangaben. Diese Angabe benötigen wir in unserem zweiten Beispiel.

## 1.4. Ein zweites Beispiel – Java Hello World

In diesem Beispiel

- 1) definieren wir zuerst ein einfaches Java Programm (HelloWorld.java)
- 2) dann übersetzen wir das Java Programm.  
Dazu benötigen wir die Task `<javac...>...</javac>` bzw. `<javac.../>`.  
Auch einen CLASSPATH müssen wir definieren.  
Zudem definieren wir das Source-Verzeichnis und das Destination-Verzeichnis (für die class-Dateien).
- 3) Jetzt können wir das Programm ausführen:  
Dazu benötigen wir die Task `<java...>..</java>` bzw. `<java.../>`  
Auch für diese Task müssen wir den CLASSPATH und eventuell benötigte Kommandozeilen-Argumente definieren.  
Aber Halt:  
das Programm kann nur ausgeführt werden, falls der Compile Step korrekt ausgeführt wurde.  
Solche Abhängigkeiten können in Ant leicht angegeben werden.

Diesmal lassen wir den XML Header unkommentiert stehen:

```
<?xml version="1.0"?>
<!-- JavaHelloWorld ant Build.xml //-->

<project default="help" basedir=". ">
  <!-- Die Task <property> erlaubt die Definition
        von "Umgebungsvariablen".
        Auf diese kann man mit ${variable} zugegriffen werden.
  -->
  <!-- flags fuer java und javac          -->
  <property name="compile.debug" value="true"/>
  <property name="compile.deprecation" value="false"/>
  <property name="compile.optimize" value="true"/>

  <!-- da stehen die Java Sourcen //-->
  <property name="src.dir" value="${basedir}/src"/>

  <!-- und hier die Class Dateien (muss angelegt werden) //-->
  <property name="bin.dir" value="${basedir}/bin"/>

  <!-- Classpath :
        Es ist besser den CLASSPATH jeweils nur pro Task,
        nicht allgemein zu setzen.
  -->
  <path id="compile.classpath">
    <!-- JAR Archive -->
    <pathelement location="${bin.dir}"/>
  </path>

  <!-- TASKS          -->
  <!-- depends=".." : diese Task fuehrt zuerst init aus
        falls die init und clean Task erfolgreich war,
        wird compile ausgefuehrt, sonst nicht.
        CLASSPATH wird direkt gesetzt
  -->
```

# ANT – DAS NEUE MAKE FÜR JAVA UND .NET

```
<target name="compile" depends="init">
  <javac srcdir="${src.dir}"
        destdir="${bin.dir}"
        classpath="${bin.dir}"
        debug="${compile.debug}" />
</target>

<target name="init" depends="clean">
  <!-- bin Verzeichnis anlegen -->
  <mkdir dir="${bin.dir}"/>
</target>

<target name="clean">
  <!-- bin Verzeichnis und Unterverzeichnisse loeschen -->
  <delete dir="${bin.dir}"/>
</target>

<target name="run" depends="compile">
  <java classname="HelloWorld" >
    <arg value="keines"/>
    <arg value="nochmal keines"/>
  <classpath>
    <!-- jars -->
    <pathelement location="${basedir}"/>
    <!-- Klassen -->
    <pathelement path="${bin.dir}"/>
  </classpath>
</java>
</target>

<target name="help">
  <echo>
    folgende Tasks sind vorhanden:
    1) uebersetzen des Java HelloWorld
       ant compile
    2) starten des Java HelloWorld
       ant run
    3) diese Ausgabe : "default" Target
       ant
    Die folgenden Tasks werden von den andern aus
    aufgerufen.
    4) clean : Target zum Loeschen der Class Dateien
       ant clean
    5) init : Target zum Anlegen des Class Dateien Dirs
       ant init
  </echo>
</target>
</project>
```

Hier einige Ausgabebeispiele:

**ant clean**

Buildfile: build.xml

clean:

[delete] Deleting directory C:\..\Ant\Beispiele\JavaHelloWorld\bin

BUILD SUCCESSFUL

Total time: 1 second

# ANT – DAS NEUE MAKE FÜR JAVA UND .NET

```
ant run
Buildfile: build.xml

clean:

init:
    [mkdir] Created dir: C:\...\Ant\Beispiele\JavaHelloWorld\bin

compile:
    [javac] Compiling 1 source file to C:\...\JavaHelloWorld\bin

run:
    [java] Wie geht's so?

BUILD SUCCESSFUL
Total time: 2 seconds
```

## *1.5.Einfache Hilfe –projecthelp*

Falls Sie sich eine Übersicht über die in einem Build.xml enthaltenen Tasks machen wollen, können Sie einfach `ant -projecthelp` aufrufen:

```
ant -projecthelp
Buildfile: build.xml
Main targets:

Subtargets:

    clean
    compile
    help
    init
    run

Default target: help
```

## 1.6. Client / Server – mehrere JVMs auf einmal

Wenn Sie eine Client Server Anwendung mit Ant verwalten wollen, haben Sie das Problem, dass mehrere JVM's gestartet werden müssen. Ant bietet dafür die Möglichkeit, pro Task eine eigene JVM zu starten: `<java... fork="true">...</java>`.

Zudem müssen die gleichzeitig auszuführenden Tasks in `<parallel>.. </parallel>` zusammengefasst werden:

```
<target name="ClientServer" description="Start Server und Client">
  <parallel>
    <java classname="xmlrpc.GetSetServer" fork="true">
      <arg line="9090"/>      <!-- port //-->
      <classpath>
        <pathelement path="${home}"/>
        <pathelement location="${xmlrpc.jar}"/>
      </classpath>
    </java>

    <!-- Server und Client parallel starten .
           fork generiert eine eigene JVM                               -->

    <java classname="Client.GetSetClient" fork="true">
      <arg line="localhost 9090 set 3456"/>
      <!-- host port get/set value //-->
      <classpath>
        <pathelement path="${home}"/>
        <pathelement location="${xmlrpc.jar}"/>
      </classpath>
    </java>
  </parallel>
</target>
```

## 1.7. Taskdef

Sie können auch eigene Ant Tasks definieren. Falls Sie die passenden Java Klassen erstellt haben, ist dies sogar recht einfach

### 1.7.1. Beschreibung

Um eine eigene Task einem aktuellen Projekt hinzuzufügen, benötigen Sie im Wesentlichen zwei Angaben:

- 1) der Name der Task: diesen können Sie frei wählen
- 2) die Java Klasse, inklusive Package, welche die Task implementiert.

Sie können sogar eine ganze Gruppe von Tasks auf einmal definieren, mithilfe eines `file` oder `resource` Attributs. Diese Attribute verweisen auf Dateien, im Java Property Format. Jede Zeile definiert eine Task.

### Format

```
taskname=fully.qualified.java.classname
```

### 1.7.2. Parameter

Attribut	Beschreibung	Muss/Kann
name	Taskname	ja, ausser "file" oder "resource" wird angegeben.
classname	Class Name (Task Klasse)	wie oben.
file	Name einer Property Datei, in der die Taskname/Classname Paare stehen	Nein
resource	Name der Property Resource, aus der Taskname/Classname Paare zu laden sind.	Nein
classpath	Classpath für <code>classname</code> oder <code>resource</code> Angaben.	Nein
classpathref	Referenz auf den Classpath für <code>classname</code> oder <code>resource</code> .	Nein
loaderRef	Loader zum Laden der spezifizierten Klasse aus dem Classpath. Sinnvoll, falls Sie mehrere Tasks auf einmal laden wollen; Tasks, die sich gegenseitig aufrufen können (ab Ant1.5)	Nein

## 1.7.3. Classpath Angabe

Das Taskdef `classpath` Attribut besitzt eine `Path` ähnliche Struktur, können also auch verschachtelt definiert werden.

## 1.7.4. Apache Axis Beispiel

Als Beispiel bietet sich Apache Axis, die neue SOAP Implementierung von Apache, an. Sie finden einige Hinweise auf Axis mit Beispielen in der Axis Starthilfe.

- 1) Installation: Entzippen in c:  
...
- 2) Nun wollen wir uns anschauen wie wir Ant Tasks definieren können

### Achtung

*Im lib Verzeichnis von Axis finden Sie das Jar `axis-ant.jar`.*

*Kopieren Sie diese Datei NICHT ins Ant Verzeichnis! Sonst haben Sie grösste Probleme, weil dann die Axis Tasks nicht gefunden werden.*

Im Archiv `axis-ant.jar` finden Sie eine Properties Datei: `axis-tasks.properties`

Diese Datei sieht folgendermassen aus:

```
#properties file for taskdefing the public Axis taskdefs  
  
axis-wsdl2java=org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask  
axis-java2wsdl=org.apache.axis.tools.ant.wsdl.Java2Wsd1AntTask  
axis-admin=org.apache.axis.tools.ant.axis.AdminClientTask
```

Die Datei definiert also drei Axis Tasks:

- 1) `axis-wsdl2java`
- 2) `axis-java2wsdl`
- 3) `axis-admin`

Neben den Namen stehen die Java Klassen, welche diese Ant Tasks implementieren.

Diese Klassen befinden sich in den Jar Dateien von Axis (siehe oben: ja nichts nach Ant kopieren!). Diese Archive stecken wir in einen Pfad:

```
<path id="axis.classpath">  
  <fileset dir="c:/axis-1_1/lib">  
    <include name="**/*.jar" />  
  </fileset>  
</path>
```

wobei ich typischerweise den absoluten Pfad auf das Axis Lib Verzeichnis in eine Property stecken würde:

```
<property name="axis.home" value="c:/axis-1_1"/>
```

Dann können wir alle physischen Abhängigkeiten zusammenfassen – das Skript lässt sich leichter neuen Gegebenheiten anpassen.

# ANT – DAS NEUE MAKE FÜR JAVA UND .NET

Nun können wir unsere Tasks definieren:

```
<taskdef resource="axis-tasks.properties" classpathref="axis.classpath" />
<!--
    #properties file for taskdefing the public Axis taskdefs
    axis-wsdl2java=org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask
    axis-java2wsdl=org.apache.axis.tools.ant.wsdl.Java2WsdlAntTask
    axis-admin=org.apache.axis.tools.ant.axis.AdminClientTask
-->

<target name="wsdl2java" description="WSDL in Java umwandeln">
    <axis-wsdl2java output="{generated.dir}"
        testcase="true" verbose="true"
        url="{local.wsdl}" >
        <mapping namespace=http://axis.apache.org/ns/interop
            package="interop"/>
    </axis-wsdl2java>
</target>

<target name="java2wsdl" description="Java in WSDL umwandeln">
    <axis-wsdl2java output="{generated.dir}"
        testcase="true" verbose="true"
        url="{local.wsdl}" >
        <mapping namespace="http://axis.apache.org/ns/interop"
            package="interop"
        />
    </axis-wsdl2java>
</target>

<target name="admin" description="Axis Administration">
    <axis-admin port="{target.port}" hostname="{target.server}"
        failonerror="true"
        servletpath="{target.appname}/services/AdminService"
        debug="true" xmlfile="{endpoint-stub.wsdd}"
    />
</target>
```

Jetzt müssen wir noch die weiteren Bestandteile, die oben referenziert werden, definieren, wie alle Verzeichnisse, Port, Host usw.

**1.7.5. Ein Anwendungsbeispiel – Apache Tomcat aus Ant aufrufen**

Damit Sie Tomcat Tasks aus Ant benutzen können, müssen Sie zuerst die Bibliothek `catalina-ant.jar` ins Verzeichnis `%ANT_HOME%/lib` kopiert haben. `ANT_HOME` ist bei mir `C:\Ant`. Das Archiv finden Sie bei `%CATALINA_HOME%/server/lib/catalina-ant.jar`. `%CATALINA_HOME%` ist das Home Verzeichnis von Apache Jakarta *Tomcat*.

Zudem ist es wichtig, dass Sie die Tasks in Ihrem `build.xml` definieren:

```
<!-- Configure props to access the Manager application -->
<property name="url" value="http://localhost:8080/manager"/>
<property name="username" value="myusername"/>
<property name="password" value="mypassword"/>

<!-- Configure the custom Ant tasks for the Manager application -->
<taskdef name="deploy"      classname="org.apache.catalina.ant.DeployTask"/>
<taskdef name="install"    classname="org.apache.catalina.ant.InstallTask"/>
<taskdef name="list"       classname="org.apache.catalina.ant.ListTask"/>
<taskdef name="reload"     classname="org.apache.catalina.ant.ReloadTask"/>
<taskdef name="remove"     classname="org.apache.catalina.ant.RemoveTask"/>
<taskdef name="resources"  classname="org.apache.catalina.ant.ResourcesTask"/>
<taskdef name="roles"      classname="org.apache.catalina.ant.RolesTask"/>
<taskdef name="start"      classname="org.apache.catalina.ant.StartTask"/>
<taskdef name="stop"       classname="org.apache.catalina.ant.StopTask"/>
<taskdef name="undeploy"   classname="org.apache.catalina.ant.UndeployTask"/>
```

**1.8. Wie geht's weiter?**

Im Ant Manual finden Sie einige Hinweise für ausgefallener Tasks. Wichtig ist das gesamte File-Handling. Aber die Grundfunktionen sollten Sie mit dieser Anleitung nutzen können.

**APACHE ANT STARTHILFE..... 1**

- 1.1. EINLEITUNG ..... 1
- 1.2. INSTALLATION..... 1
- 1.3. EIN ERSTES BEISPIEL – HELLOWORLD AUSGABE..... 2
- 1.4. EIN ZWEITES BEISPIEL – JAVA HELLOWORLD..... 3
- 1.5. EINFACHE HILFE –PROJECTHELP ..... 5
- 1.6. CLIENT / SERVER – MEHRERE JVMs AUF EINMAL ..... 6
- 1.7. TASKDEF ..... 7
  - 1.7.1. *Beschreibung*..... 7
  - 1.7.2. *Parameter*..... 7
  - 1.7.3. *Classpath Angabe*..... 8
  - 1.7.4. *Apache Axis Beispiel*..... 8
  - 1.7.5. *Ein Anwendungsbeispiel – Apache Tomcat aus Ant aufrufen*..... 10
- 1.8. WIE GEHT'S WEITER? ..... 10