

18 Templates

Ant-Kommandos sind aufgrund der XML-Syntax und der Vielzahl von Optionen recht umfangreich. Seit der Version 1.6 bietet Ant daher die Möglichkeit, unterschiedliche Templates zu erstellen, in die zur Laufzeit dynamisch Parameter eingesetzt werden.

18.1 Presetdef

Das Kommando `<presetdef>` ermöglicht es Ihnen, eigene Definitionen für Tasks und Datentypen zu schaffen. Die neuen Definitionen basieren auf den vorhandenen Kommandos und Datentypen, wobei Sie Defaultwerte für Attribute und Sub-Tags definieren können. Wenn Sie diese neuen Definitionen später verwenden, so werden die vordefinierten Einstellungen benutzt. Sie können diese Einstellungen aber überschreiben bzw. ergänzen.

Das folgende Beispiel zeigt eine Definition für einen Compiler-Task `my.javac`, bei dem einige wichtige allgemeine Einstellungen schon gesetzt sind.

```
<project name="bsp1801" default="main">
  <property name="path.srcdir"
    value="../kap21/java/src"/>
  <property name="path.builddir"
    value="../kap21/java/build"/>
  <presetdef name="my.javac">
    <javac fork="yes"
      memoryinitialsize="32m"
      memorymaximumsize="512m"
      debug="true"
      srcdir="${path.srcdir}"
      destdir="${path.builddir}"
      optimize="true"
      compiler="modern">
    </javac>
  </presetdef>
```

```

<target name="main">
  <mkdir dir="{path.builddir}" />
  <my.javac includes="dialogs/CheckboxDialog*" />
  <my.javac includes="extensions/*" />
</target>
</project>

```

Diesen neuen Task können Sie später benutzen, um unterschiedliche Zweige aus dem Quellverzeichnis zu kompilieren. Als einzige Angabe ist dabei das Attribut `includes` notwendig. Alle anderen Attribute entnimmt Ant aus dem Template. Der Einsatz von `<presetdef>` bietet sich an, um Schreibearbeit zu sparen.

Das Beispiel kompiliert einige Java-Dateien, die im Kapitel 21 (Ant erweitern durch Java-Programmierung) näher erläutert werden.

18.2 Macrodef

Interessanter als die Vorbelegung von Tasks und Datentypen mit Attributen oder Sub-Tags ist das Schreiben komplexer Makros mittels des Kommandos `<macrodef>`. Mit diesem Kommando können Sie mehrere Tasks und Datentypen zu einer größeren Einheit zusammenfassen, die in etwa einem Target entspricht.

Der Aufruf eines Makros erfolgt direkt über ein XML-Tag mit dem Namen des Makros. Sie können dabei Attribute und Sub-Tags übergeben, falls das Makro dies gestattet. Die an ein Makro übergebenen Attribute sind übrigens keine Property's im herkömmlichen Sinne und müssen im Makro anders behandelt werden.

Der `<macrodef>`-Task erfordert auf jeden Fall das Attribut `name`. Über den Wert dieses Attributs wird das Makro später angesprochen. Innerhalb des Makros können Sie 4 Sub-Tags verwenden. Alle Ant-Tasks, deren Aufruf Sie direkt im Makro programmieren möchten, müssen Sie im Makro durch ein `<sequential>`-Tag zusammenfassen. Falls Sie beim Aufruf des Makros zusätzliche Sub-Tags übergeben möchten, müssen Sie im Makro mit dem Sub-Tag `<element>` entsprechende Platzhalter definieren. Zur Übernahme von Attributen dient das Sub-Tag `<attribute>`. Innerhalb des Makros sprechen Sie die Attribute durch den Platzhalter

```
@{attributname}
```

an. Die Attribute der Makros dürfen nicht mit den herkömmlichen Property's verwechselt werden.

Falls beim Aufruf des Makros auch der XML-Textkörper des Tags übergeben werden soll, muss dieser im Makro durch das Sub-Tag `<text>` übernommen werden.

Im Gegensatz zu den per `<ant>` oder `<antcall>` gerufenen Targets wird für ein Makro kein separater Datenkontext angelegt. Wenn Sie innerhalb des Makros Property's definieren, dann existieren diese Property's anschließend auch außerhalb des Makros.

Die Attribute der beiden Sub-Tags `<attribute>` und `<element>` finden Sie in Tabelle 18–1. Die andern beiden Sub-Tags besitzen keine Attribute und werden deshalb nicht in die Tabelle aufgenommen.

<makrodef>				
Sub-Tag	Attribut	Beschreibung	Default	Erforderlich
<code><attribute></code>	<code>name</code>	Name des Attributs		Ja
	<code>default</code>	Defaultwert		Nein
<code><element></code>	<code>name</code>	Name des Platzhalters		Ja
	<code>optional</code>	Platzhalter ist optional:	<code>false</code>	Nein
	<code>implicit</code>	Sub-Tags des Makroaufrufs einfügen.	<code>false</code>	Nein

Tab. 18–1 Attribute der Sub-Tags des `<makrodef>`-Kommandos

Die Definition und die Anwendung eines Makros sind relativ einfach. Als Beispiel soll die Ausgabe einiger Debug-Informationen dienen.

Das erste Makro benutzt die Parameterübergabe per Attribut, um ausgewählte Property's aufzulisten, sowie ein implizites Element, um eine Überschrift auszugeben.

```
<project name="bsp1802" default="main">
  <macrodef name="echo.prop">
    <attribute name="prefix" default="" />
    <text name="header" trim="yes" optional="yes"/>

    <sequential>
      <echo message="@{header}"/>
      <echoproperties prefix="@{prefix}"/>
    </sequential>
  </macrodef>

  <target name="main">
    <echo.prop prefix = "file">
      Alle Properties mit Prefix 'file'
    </echo.prop>
  </target>
</project>
```

Im Makro wird mit dem `<attribute>`-Sub-Tag ein Parameter `prefix` definiert. Dieser Parameter dient später im `<echoproperties>`-Tag zur Auswahl der aufzulistenden Property's. Beachten Sie dabei bitte den Zugriff auf den Attributwert mittels des `@{...}`-Platzhalters.

Per `<text>`-Sub-Tag definiert das Makro einen Platzhalter für die Textbestandteile des aufrufenden Tasks. Der Text ist dann ebenfalls über ein Attribut, hier mit dem Namen `header`, zugänglich.

In der Anwendung wird das Makro durch ein Tag aufgerufen, das dem Namen des Makros entspricht, also `echo.prop`. Es gibt keine expliziten Tasks, mit denen ein Makro ausgeführt wird. Dies vereinfacht zwar die Notation der Build-Datei, verursacht aber Fehler, wenn die Build-Datei mittels einer DTD validiert wird.

Im Beispiel wird dem Makro beim Aufruf das Attribut `prefix` übergeben. Der Name muss mit einem der im Makro per `<attribute>`-Sub-Tag definierten Attribute übereinstimmen. Weiterhin verfügt das aufrufende Tag über ein Textelement. Dieses wird als Überschrift benutzt.

Das zweite Beispiel demonstriert das Einfügen von Elementen, also kompletten Codeteilen.

```
<project name="bsp1803" default="main">
  <macrodef name="echo.debug">
    <element name="echo.header"/>
    <element name="echo.content"/>
    <sequential>
      <echo.header/>
      <echo.content/>
    </sequential>
  </macrodef>

  <target name="main">
    <echo.debug>
      <echo.content>
        <concat>
          <fileset dir=".">
            <include name="*xml"/>
          </fileset>
        </concat>
      </echo.content>
      <echo.header>
        <echo>Dateiininhalt</echo>
      </echo.header>
    </echo.debug>
  </target>
</project>
```

Dazu müssen Sie im Makro zunächst per `<element>`-Tag Namen für die einzufügenden Abschnitte definieren. In diesem Beispiel sind das `echo.header` und `echo.content`. Innerhalb des `<sequential>`-Sub-Tags werden die Platzhalter in der gewünschten Reihenfolge notiert. Sie werden später durch die echten Anweisungen aus dem rufenden Task ersetzt.

Beim Aufruf werden die Namen der Elemente-Platzhalter als eigenständiges Tag notiert. Alle Anweisungen, die zwischen dem öffnenden und dem schließenden Tag stehen, werden später im Makro an der vorgegebenen Stelle eingesetzt.

Ein ähnlich funktionierendes Tag ist `<scriptdef>`, das im Kapitel 20 (Scripting) detaillierter beschrieben wird.