Wie konfiguiriert man Eclipse (mit oder ohne Plugin)

Erich Ehses

Man kann die nötigen Dateien separat von den angegebenen Quellen beziehen oder das Eclipse-Plugin für java_cup verwenden. Am Ende benötigt man drei Archiv-Dateien:

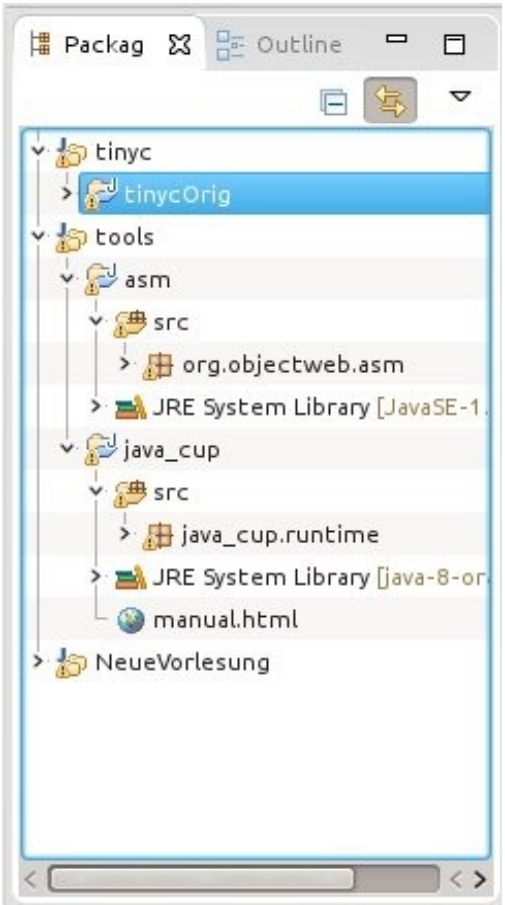
jflex.jar	für die lexikalische Spezifikation
java_cup.jar	für die Grammatik-Spezifikation
java_cup_runtime.jar	Laufzeitklassen

Der Vorteil des Plugins ist der Cup-Editor und die „vorgekauten“ Templates. Der Editor funktioniert aber nur mit Java8. Bei jflex gibt es keine Unterstützung.

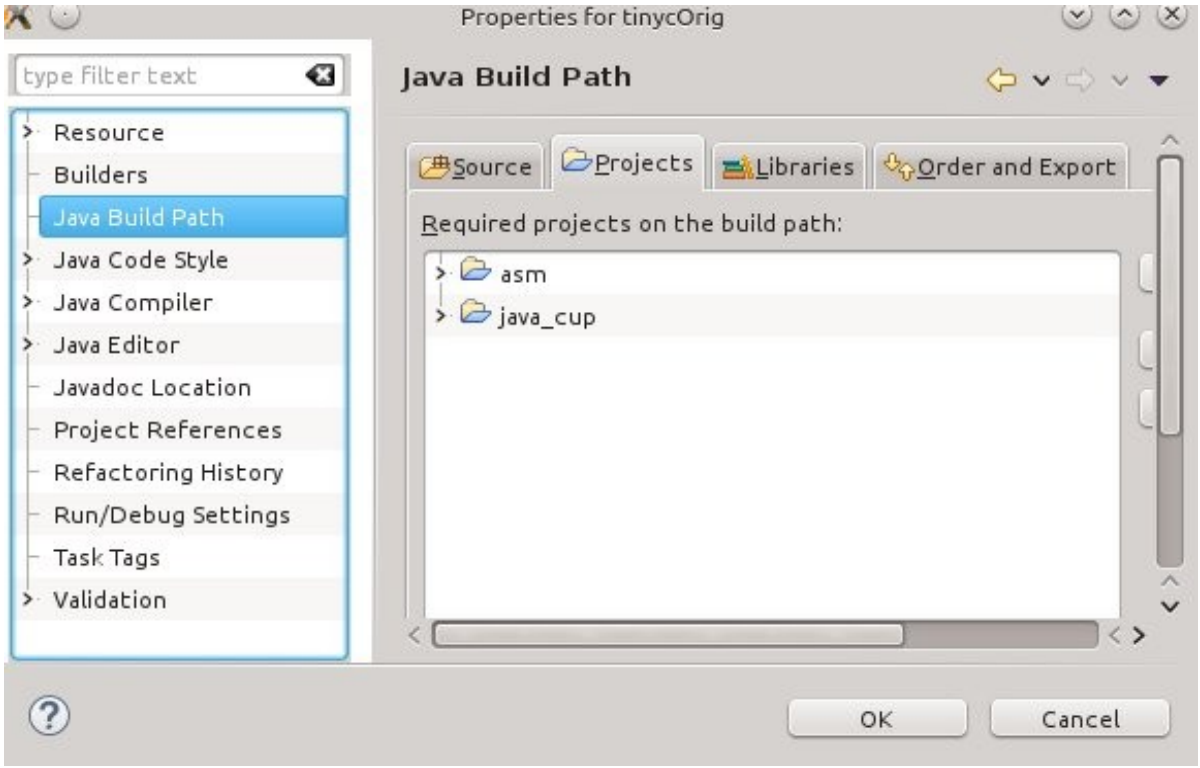
Der zweite wichtige Punkt ist der Buildprozess. Hierzu liefert das Plugin vorkonfigurierte Projekte mit vorkonfiguriertem build.xml. Letztere müssen aber in aller Regel angepasst werden: für den Einstieg ist die diese Variante gut geeignet.

Bei komplexer gestalteten Projekten sollte man aber wissen wie die Zusammenhänge sind und wie man alles „zu Fuß“ macht.

Zunächst ist es wichtig, dass das Projekt weiß, wo sich die nötigen Bibliotheken befinden. Bei mir wird außer java_cup_runtime.jar auch arm.jar für die Generierung von Java-Classfiles benötigt. Ich habe das so gemacht, dass beide Bibliotheken eigene Projekte innerhalb meines Eclipse-Workspace darstellen:

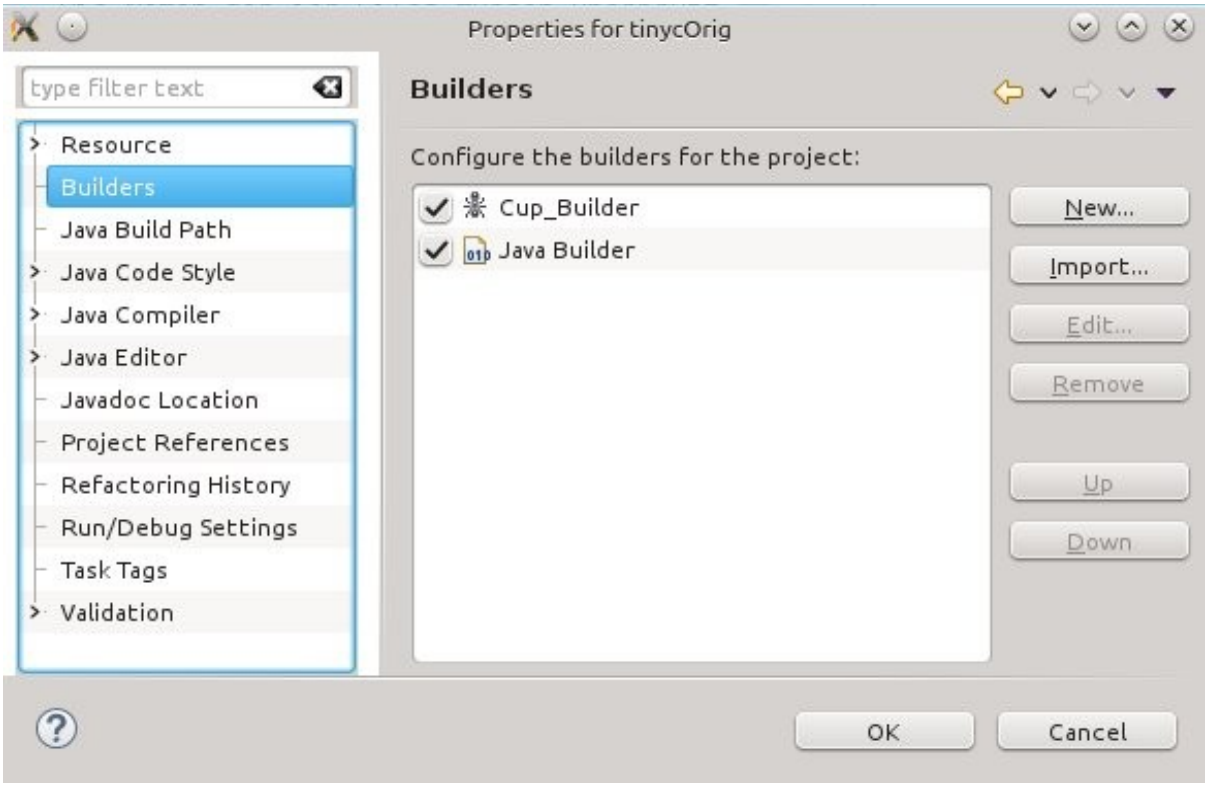


Jetzt muss man dem Projekt noch sagen, dass die Projekte in den CLASSPATH aufgenommen werden sollen:



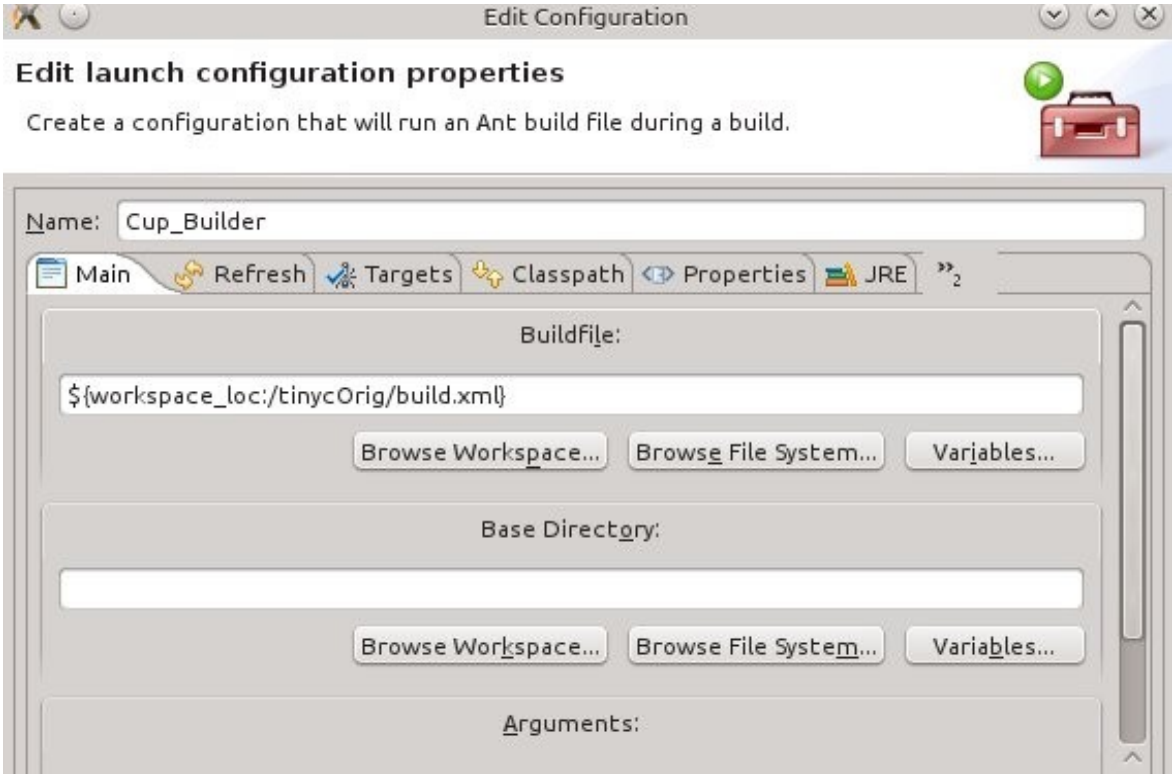
Natürlich sieht das anders aus, wenn sich die Bibliotheken woanders befinden. Das könnte dann unter „Libraries“ der Verweis auf „external jars“ sein.

Der Buildprozess wird in „Properties → Builders“ beschrieben. Er setzt voraus, das es im Projekt eine „build.xml“-Datei gibt, die die Aktionen und Abhängigkeiten zum Generieren der Syntax-Klassen beschreibt. Dazu kommen wir noch. Zunächst muss man einen neuen Builder erzeugen (falls es ihn nicht schon gibt):



Es ist darauf zu achten, dass der neue Builder (hier „Cup_Builder“) vor dem Java Builder steht. Nur dann werden automatisch die richtigen Aktionen ausgeführt.

Jetzt ist nur noch einzutragen, wo sich die Datei build.xml befindet. Anstelle von tinycOrig sollte dann der korrekte Projektename stehen.



Wenn man alles richtig konfiguriert hat, sollte man keine Unterschied zu dem normalen „Komfort“ von Eclipse erkennen.

Schließlich ist noch die Datei „build.xml“ zu Erzeugen oder abzuändern. Die Kommentare sollten ausreichen. Eigentlich sind auch nur Pfade und eventuell auch ein paar Optionen anzupassen. Wenn man auf die Unterstützung für die Erzeugung eines ausführbaren jar-Files keinen Wert legt, wird die Datei nochmal deutlich einfacher. WICHTIG: im Zusammenhang mit der Builder-Einstellung ist unbedingt dein „Defaulttarget“ anzugeben!

```
<project name="tinyc" default="scanner" basedir=".">
      <description>Build for tinyc-project</description>

<!--
      Der folgende Pfad muss unbedingt angepasst werden!
      Die Namen der jar-Files müssen überprüft werden!
      Der Rest passt auf das tinyc-Projekt. Bei anderen Projekten muss
      das natürlich auch geändert werden.
-->

    <!-- lib: Ort von java_cup, jflex -->
    <property name ="lib" location = "/home/erich/Lehre/compiler/lib" />

    <!-- Referenz zu den Ant-Tasks von jflex und javacup-->
    <taskdef name="jflex"          classname="JFlex.anttask.JFlexTask"
        classpath="${lib}/JFlex.jar" />
```

```
    <taskdef name="cup"            classname="java_cup.anttask.CUPTask"
        classpath="${lib}/cup.jar" />

    <!-- src: Ort der Deklarationsdateien (.cup, .jflex) -->
    <property name ="src"          location="./src/parser" />

    <!-- Namen der Deklarationsdateien -->
    <property name ="cup_name"     value="tinyc.cup" />
    <property name ="lex_name"     value="tinyc.flex" />

    <!-- gen: Ort der generierten Dateien -->
    <property name ="gen" location="./src" />

    <!-- Namen der generierten Dateien -->
    <property name ="parser_name"  value="Parser" />
    <property name ="sym_name"     value="Symbols" />
    <property name ="scanner_name" value="Scanner" />

    <!-- nur für die Erzeugung von jar-Files -->
    <property name ="main_class"   value = "parser.TinyC"/>
    <property name ="classes"      location="./bin" />
    <property name ="jar_file"     location= "/home/erich/tinyc.jar" />
    <property name ="runtime"      location= "../java_cup/bin" />
    <property name ="asm"          location= "../asm/bin" />

    <!-- Generierung des Parsers, mit Optionen -->
    <target name="parser">
        <cup srcfile="${src}/${cup_name}" destdir="${gen}"
            package="gener"
            parser="${parser_name}"
            symbols="${sym_name}"
            nopositions="true"
            interface="false"/>
    </target>

    <!-- Generierung des Scanners, mit Optionen -->
    <target name="scanner" depends="parser">
        <jflex file="${src}/${lex_name}" destdir="${gen}"
            nobak="true"/>
    </target>

      <!-- Der Rest ist für die jar-File Erzeugung -->
      <target name = "compile" depends = "scanner">
          <javac srcdir="${gen}"
              destdir="${classes}"
              classpath="${runtime}"
              includeantruntime="false"/>
      </target>

      <target name="jar" depends = "compile">
          <jar destfile="${jar_file}" >
              <fileset dir = "${classes}" />
              <fileset dir = "${runtime}" />
              <fileset dir = "${asm}" />
              <manifest>
                   <attribute name="Main-Class" value = "${main_class}"/>
              </manifest>
          </jar>
      </target>
</project>
```

Man sollte auch ruhig mal den Buildfile des Plugins zu Vergleich heranziehen:

```
<project name="MyProject" default="compile" basedir=".">

        <property name="java"     location="src"       />
        <property name="classes"  location="bin/cls"   />
        <property name="result"   location="bin/jar"   />
        <property name="lib"      location="lib"       />
        <property name="tools"    location="tools"     />
        <property name="base"    location="."     />

        <taskdef name="jflex" classname="JFlex.anttask.JFlexTask"  classpath="${tools}/JFlex.jar"   />
        <taskdef name="cup"   classname="java_cup.anttask.CUPTask" classpath="${tools}/java-cup-11b.jar"  />

        <target name="generate">

          <cup srcfile="${base}/parser.cup"  destdir="${java}"
               parser="Parser"
                   interface="true"
                 locations="true"
               debugsymbols="true" />
            <jflex  file="lexer.jflex" destdir="${java}" />
        </target>

        <path id="libraries">
            <files includes="${tools}/java-cup-11b-runtime.jar" />
        </path>

        <target name="compile" depends="generate">
            <mkdir dir="${classes}"/>
            <mkdir dir="${result}"/>

          <javac includeantruntime="false" srcdir="${java}" destdir="${classes}">
            <classpath refid="libraries" />
          </javac>
        </target>

        <target name="clean">
          <delete file="${java}/Parser.java" />
          <delete file="${java}/sym.java" />
          <delete file="${java}/Scanner.java" />
          <delete dir="${classes}" />
          <delete dir="${result}" />
        </target>
</project>
```