How to use ant with ANTLR3

How to use ant with ANTLR3 ?

Verify that all prerequisites are fulfilled

1. Make sure antlr2, antlr3, ST are in classpath. All in one is here:

   http://antlr.org/download/antlr-3.3-complete.jar

2. For a quick check if ANTLR3 can be found via classpath, copy the following lines into a file named build.xml:

```
<project name="antlr3_and_ant" default="antlr_classpath" basedir=".">

    <!-- Check if ANTLR3 can be found in the classpath -->
    <target name="antlr_classpath">
        <whichresource property="antlr.in.classpath" class="org.antlr.Tool" />
        <fail message="ANTLR3 not found via CLASSPATH ${java.class.path}">
            <condition>
                <not>
                    <isset property="antlr.in.classpath"/>
                </not>
            </condition>
        </fail>
        <echo>ANTLR3 found via CLASSPATH</echo>
    </target>

</project>
```

   Execute ant in the directory you saved the build.xml file in:

```
$ ant
```

   The ant command should result in some output similar to that shown below:

```
Buildfile: F:/tmp/build.xml

antlr_classpath:
     [echo] ANTLR3 found via CLASSPATH

BUILD SUCCESSFUL
Total time: 0 seconds
```

In case ANTLR3 is not contained in the class path, the build will fail. To convince you that ANLTR3 is not part of the CLASSPATH, the complete CLASSPATH visible to ant is printed.

```
Buildfile: F:/tmp/build.xml

antlr_classpath:

BUILD FAILED
F:/tmp/build.xml:6: ANTLR3 not found via CLASSPATH
F:/apache-ant-1.8.2/lib/ant-launcher.jar;F:\apache-ant-1.8.2\lib\ant-antlr.jar;...;C:\Programme
time: 0 seconds
```

In summary, what we have done up to this point:

- Verified that ant is installed and working
- Proved that ANTLR3 is contained in the classpath

Now, we are ready to let ant compile the first ANTLR3 grammar.

A first example using the Java-task

In the following sections the grammar CMinus.g is used. The grammar is part of the ANTLR V3 sample grammars archive and can be downloaded from here: http://www.antlr.org/download/examples-v3.tar.gz

The ant Java-task will be used to let ANTLR3 compile a grammar. As you can see, it is mainly a mapping of the command-line onto the syntax of the ant Java-task:

```
$ java org.antlr.Tool -verbose -o . CMinus.g
```

The order of command-line arguments must match ANLTR3s command line syntax.

Save the few XML lines below in a file named build.xml in the CMinus example directory.

```
<project name="cminus" default="antlr" basedir=".">

    <!-- Antlr3 is called here -->
    <!-- java org.antlr.Tool -verbose -o . CMinus.g -->
    <target name="antlr">
       <java classname="org.antlr.Tool" fork="true" failonerror="true">
          <arg value="-verbose"/>
          <arg value="-o"/>
          <arg path="."/>
          <arg path="CMinus.g"/>
       </java>
    </target>

</project>
```

All ANTLR3 command-line options which refer to a directory or file path (e.g. -o or -lib) should use the path attribute within the embedded <argv> element. This means to break up a sequence like

```
"-o /home/dev/examples/cminus"
```

into:

```
<arg value="-o"/>
<arg path="/home/dev/examples/cminus"/>
```

The path attribute will convert the path to the platform's local conventions.

As the default task is defined as "antlr" in the project statement, just type on the command-line to let ANTLR3 compile the grammar CMinus.g:

```
$ ant
```

This should give the following result:

```
Buildfile: F:\examples-v3\java\cminus\build.xml

antlr:
     [java] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
     [java] F:\examples-v3\java\cminus\CMinus.g

BUILD SUCCESSFUL
Total time: 2 seconds
```

The available ANTLR3 command-line options can be found here http://www.antlr.org/wiki/display/ANTLR3/Command+line+options or by calling ANTLR3 from the command-line without arguments:

```
$ java org.antlr.Tool
```

The next step is to compile the ANLTR3 generated Java-files. We already assured that ANTLR3 is contained in the CLASSPATH in the last chapter.

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

    <!-- Antlr3 is called here -->
    <!-- java org.antlr.Tool -verbose -make -o . CMinus.g -->
    <target name="antlr">
       <java classname="org.antlr.Tool" fork="true" failonerror="true">
          <arg value="-verbose"/>
          <arg value="-make"/>
          <arg value="-o"/>
          <arg path="."/>
          <arg path="CMinus.g"/>
       </java>
    </target>

    <target name="compile" depends="antlr" description="compile">
       <javac srcdir="."
              destdir="."
              deprecation="Yes"
              listfiles="Yes"
              includeantruntime="false">
          <classpath>
              <pathelement path="${java.class.path}"/>
          </classpath>
        </javac>
    </target>

</project>
```

The output generated should be similar to this:

```
Buildfile: F:\examples-v3\java\cminus\build.xml

antlr:
     [java] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
     [java] F:\examples-v3\java\cminus\CMinus.g

compile:
    [javac] Compiling 2 source files to F:\examples-v3\java\cminus
    [javac] F:\examples-v3\java\cminus\CMinusLexer.java
    [javac] F:\examples-v3\java\cminus\CMinusParser.java
    [javac] Note: F:\examples-v3\java\cminus\CMinusParser.java uses unchecked or unsafe
operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

BUILD SUCCESSFUL
Total time: 6 seconds
```

The ANLTR3 command-line option "-make" only generates new files in case they are older than the grammar.
This behaviour might have an effect on dependent tasks like "compile", which could result in nothing to be processed because it is up to date.

A second execution of the build-file shows that nothing is built by ANLTR3 and therefore nothing has to be recompiled:

```
Buildfile: F:\examples-v3\java\cminus\cminus.xml

antlr:
     [java] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
     [java] Grammar F:\examples-v3\java\cminus\CMinus.g is up to date - build skipped

compile:

BUILD SUCCESSFUL
Total time: 1 second
```

A more complex build with multiple dependent grammars

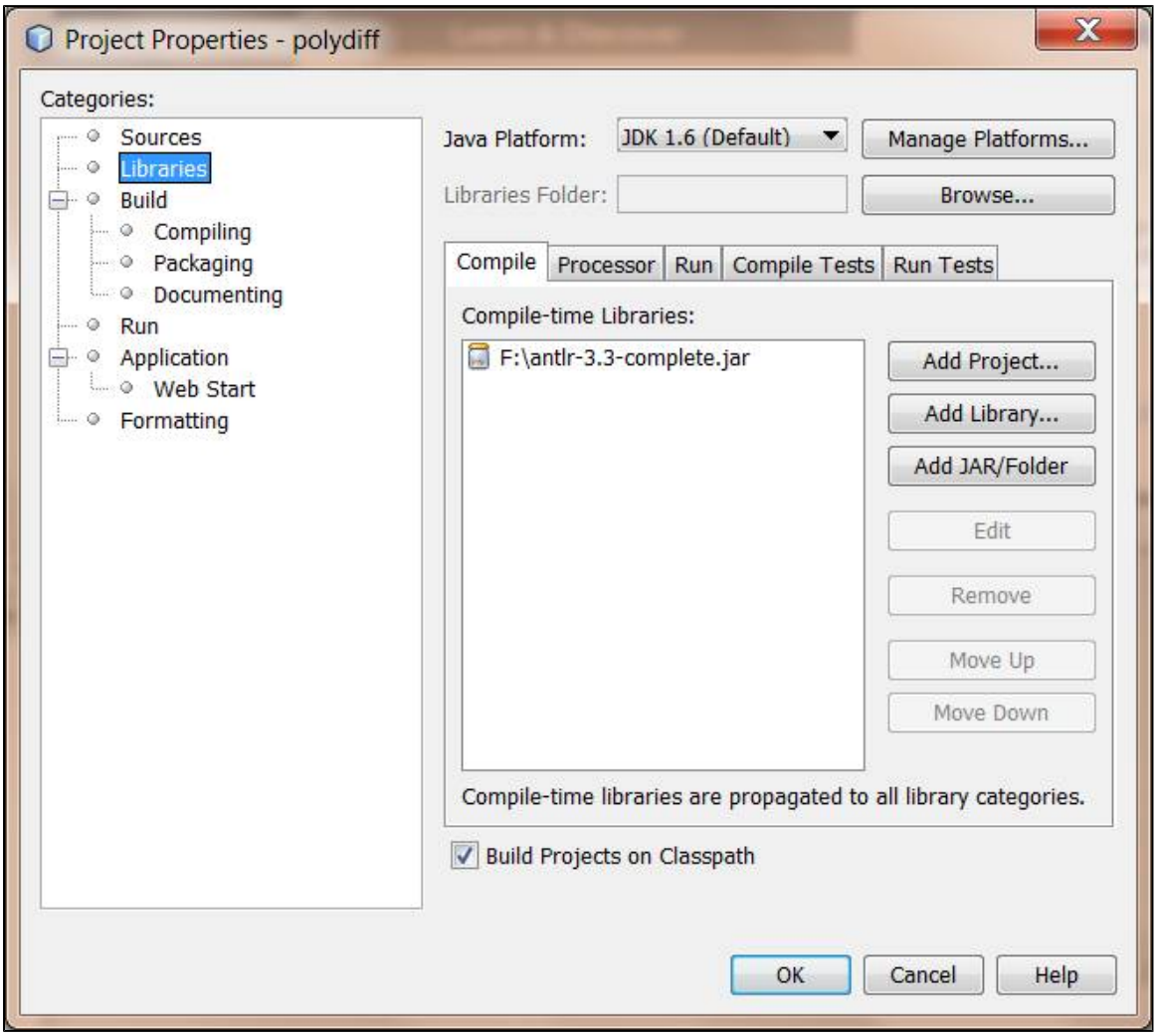
The Polydiff example is also part of the ANTLR V3 sample grammars archive. This example consists of four dependent grammars.

The ANTLR3 make option will be used to decide if a recompilation of the grammar is necessary.

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

   <!-- An ant macro which invokes ANTLR3
         This is just a parameterizable wrapper to simplify the invocation of ANTLR3.
         The default values can be overriden by assigning a value to an attribute
         when using the macro.
         Example with ANTLR3 outputdirectory modified:
            <antlr3 grammar.name="CMinus.g" outputdirectory="${src}/${package}"/>
    -->
    <macrodef name="antlr3">
        <attribute name="grammar.name"/>
        <attribute name="outputdirectory" default="."/>
        <attribute name="libdirectory" default="."/>
        <sequential>
        <java classname="org.antlr.Tool" fork="true" failonerror="true">
           <arg value="-o"/>
           <arg path="@{outputdirectory}"/>
           <arg value="-lib"/>
           <arg path="@{libdirectory}"/>
           <arg value="-verbose"/>
           <arg value="-Xmultithreaded"/>
           <arg value="-make"/>
           <arg path="@{grammar.name}"/>
        </java>
        </sequential>
    </macrodef>

    <target name="PolyPrinter" depends="Simplifier">
       <antlr3 grammar.name="PolyPrinter.g" />
    </target>
    <target name="Simplifier" depends="PolyDifferentiator">
        <antlr3 grammar.name="Simplifier.g" />
    </target>
    <target name="PolyDifferentiator" depends="Poly">
        <antlr3 grammar.name="PolyDifferentiator.g" />
    </target>
    <target name="Poly">
        <antlr3 grammar.name="Poly.g" />
    </target>

    <target name="compile" depends="PolyPrinter" description="compile">
        <javac srcdir="." destdir="."
               listfiles="Yes" deprecation="Yes">
           <compilerarg value="-Xlint:unchecked"/>
           <classpath>
               <pathelement path="${java.class.path}"/>
           </classpath>
        </javac>
    </target>

</project>
```

ANTLR3 and Netbeans

To make the ANTLR3 libraries available to the Netbeans classpath, use the project properties dialog:



Add the ANTLR3 libraries to the Compile- and the Run-Path.

The ANTLR3 libraries can be referenced in the netbeans build file via the ${javac.classpath} and the ${run.classpath} properties.

```
    ...
    <!-- where to place the antlr generated files -->
    <property name="src" location="src"/>
    <!-- name of the package -->
    <property name="package" value=""/>
    <!-- where to find the grammar files -->
    <property name="grammar" location="grammar"/>
    <!-- where to write/find token files -->
    <property name="token.lib" location="${src}/${package}" />

    <macrodef name="antlr3">
       <attribute name="grammar.name"/>
       <attribute name="package" default="${package}"/>
       <sequential>
       <echo message="antlr ${grammar}/@{grammar.name}" />
       <!-- Antlr3 is called here -->
       <java classname="org.antlr.Tool" fork="true" failonerror="true">
          <jvmarg value="-Xmx512M"/>
          <arg value="-o"/>
          <arg path="${src}/${package}"/>
          <arg value="-lib"/>
          <arg path="${src}/${package}"/>
          <arg value="-verbose"/>
          <arg value="-report"/>
          <arg value="-Xmultithreaded"/>
          <arg value="-make"/>
          <arg path="${grammar}/@{grammar.name}"/>
          <classpath>
               <pathelement path="${run.classpath}"/>
           </classpath>
       </java>
       </sequential>
    </macrodef>

    <!-- Modify here. Adapt the dependencies for your grammar(s).
         Watch out to specify the dependencies in the correct order!
         The sequence in which the grammars are compiled by Antlr in
         this example:
         1: Poly.g as PolyDifferentiator depends on it
         2: PolyDifferentiator as Simplifier dependends on it
         3: Simplifier.g as PolyPrinter dependends on it
         4: PolyPrinter.g
    -->
    <target name="-pre-compile" depends="PolyPrinter"/>
    <target name="PolyPrinter" depends="Simplifier">
        <antlr3 grammar.name="PolyPrinter.g"/>
    </target>
    <target name="Simplifier" depends="PolyDifferentiator">
        <antlr3 grammar.name="Simplifier.g"/>
    </target>
    <target name="PolyDifferentiator" depends="Poly">
        <antlr3 grammar.name="PolyDifferentiator.g"/>
    </target>
    <target name="Poly">
        <antlr3 grammar.name="Poly.g"/>
    </target>

</project>
```

ANTLR3 task for ant

The ANTLR3 task for ant is another choice to invoke ANTLR3 which can be used instead of the Java-task or Exec-task in ant.

It seamlessly integrates in the ant build process and supports the ant syntax.

The ANTLR3-task is implicitely evaluating dependencies via ANTLR3s depend option. This makes a difference when multiple grammar files are involved.

⚠ The ANTLR3 task for ant is available for ANTLR3 V3.2 and later.

Installation of the ANTLR3 task for ant

1. Download ant-antlr3 task from

   http://www.antlr.org/share/1169924912745/antlr3-task.zip

2. Copy the antlr3-task.zip file to a convenient temporary location. e.g /tmp and unpack the archive

   ```
   $unzip antlr3-task.zip
   ```

   The directory structure unveiled should look like this:

   ```
   directory       antlr3-src              'source code of the utility
   directory       examples                'some example build files on how to use the
   ant-antlr3 task
   file            ant-antlr3.jar          'the ant task for antlr3
   file            antlr3-task.doc         'some short documentation (Word-Document)
   file            antlr3-task.htm         'some short documentation (HTML-Document)
   file            Readme.txt              'history of changes and a few hints
   ```

3. Make **ant-antlr3.jar** visible to ant. The recommended way to achieve this, is to copy the **ant-antlr3.jar** into your $ANT_HOME/lib directory.

   Assuming apache-ant-1.8.2 is installed in /usr/local, you might proceed as shown below:

   ```
   $ sudo cp /tmp/ant-antlr3.jar /usr/local/apache-ant-1.8.2/lib/
   ```

   Check that the antlib **ant-antlr3.jar** is correctly recognized by ant:

   ```
   $ ant -diagnostics
   ```

   should show:

   ```
   \----------------------------------------\-
   ANT_HOME/lib jar listing
   \----------------------------------------\-
   ant.home: /usr/share/ant
   ant-antlr.jar (5758 bytes)
   ant-antlr3.jar (20.889 bytes)      <--------\-
   ...
   ```

   Make sure that antlr-3.3-complete.jar is contained in the classpath. The same prerequisites have to be fulfilled and should be checked as described before.

A first example using the ANTLR3 task

In this section the grammar CMinus.g is used.

The ANTLR3 task for ant will be used to let ANTLR3 compile a grammar. The order of arguments does not matter, which is not true when you use the Java-task or Exec-task.

Save the few XML lines below in a file named build.xml in the CMinus example directory.

```
<project name="cminus" default="antlr" basedir=".">

    <!-- Antlr3 is called here -->
    <target name="antlr">
        <antlr:ant-antlr3 xmlns:antlr="antlib:org/apache/tools/ant/antlr"
            verbose="true"
            outputdirectory="."
            target="CMinus.g"/>
    </target>

</project>
```

As the default task is defined as "antlr" in the project statement, to let ANTLR3 compile the grammar CMinus.g type on the command-line:

```
$ ant
```

This should give the following result:

```
Buildfile: F:\examples-v3\java\cminus\build.xml

antlr:
[antlr:ant-antlr3] F:\examples-v3\Java\cminus\CMinus.g
[antlr:ant-antlr3] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56

BUILD SUCCESSFUL
Total time: 3 seconds
```

The next step is to compile the ANLTR3 generated Java-files. We already assured that ANTLR3 is contained in the CLASSPATH in the last chapter.

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

    <!-- Antlr3 is called here -->
    <target name="antlr">
       <antlr:ant-antlr3 xmlns:antlr="antlib:org/apache/tools/ant/antlr"
            verbose="true"
            outputdirectory="."
            target="CMinus.g"/>
    </target>

    <target name="compile" depends="antlr" description="compile">
       <javac srcdir="."
              destdir="."
              deprecation="Yes"
              listfiles="Yes"
              includeantruntime="false">
           <classpath>
               <pathelement path="${java.class.path}"/>
           </classpath>
        </javac>
    </target>

</project>
```

The output generated should be similar to this:

```
Buildfile: F:\examples-v3\java\cminus\build.xml

antlr:
[antlr:ant-antlr3] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
[antlr:ant-antlr3] F:\examples-v3\Java\cminus\CMinus.g

compile:
    [javac] Warning: CMinus.tokens modified in the future.
    [javac] Warning: CMinusLexer.java modified in the future.
    [javac] Compiling 2 source files to F:\examples-v3\Java\cminus
    [javac] F:\examples-v3\Java\cminus\CMinusLexer.java
    [javac] F:\examples-v3\Java\cminus\CMinusParser.java
    [javac] Note: F:\examples-v3\Java\cminus\CMinusParser.java uses unchecked or unsafe
operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

BUILD SUCCESSFUL
Total time: 9 seconds
```

How does the ANTLR3 task for ant handle multiple grammar files

The Polydiff example is also part of the ANTLR V3 sample grammars archive. This example consists of four dependent grammars. The ANTLR3 task for ant will use ANLTR3s depend option to receive a list of affected files. The time stamps of the files delivered are used by the ANTLR3 task for ant to decide if a recompile of a grammar is necessary.

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

    <!-- An ant macro which invokes antlr3
         This is just a parameterizable wrapper to simplify the invocation of ANTLR3.
         The default values can be overriden by assigning a value to an attribute
         when using the macro.
         Example with ANTLR3 debug option activated and the outputdirectory modified:
            <antlr3 grammar.name="CMinus.g" outputdirectory="${src}/${package}" debug="true"/>
    -->
    <macrodef name="antlr3">
        <attribute name="grammar.name"/>
        <attribute name="outputdirectory" default="."/>
        <attribute name="libdirectory" default="."/>
 <attribute name="multithreaded" default="true"/>
 <attribute name="verbose" default="true"/>
 <attribute name="report" default="true"/>
        <attribute name="debug" default="false"/>
        <sequential>
            <antlr:ant-antlr3 xmlns:antlr="antlib:org/apache/tools/ant/antlr"
                target="@{grammar.name}"
                outputdirectory="@{outputdirectory}"
                libdirectory="@{libdirectory}"
                multithreaded="@{multithreaded}"
                verbose="@{verbose}"
                report="@{report}"
                debug="@{debug}">
                <classpath>
                    <pathelement path="${java.class.path}"/>
                </classpath>
                <jvmarg value="-Xmx512M"/>
            </antlr:ant-antlr3>
        </sequential>
    </macrodef>

    <!-- ANTLR3 is called here -->
    <!-- The dependency among grammars has to be defined by the invocation sequence  -->
    <!-- The hierarchy in this example is:

         Poly.g
```

```
     - PolyDifferentiator.g
        - Simplifier.g
           - PolyPrinter.g

     The dependency is described from bottom to top:

     PolyPrinter.g depends from
     - Simplifier.g depends from
        - PolyDifferntiator.g depends from
           - Poly.g

     Note that Antlr is only invoked when necessary, e.g.
     changes in PolyDifferentiator.g do not result in
     Poly.g being recompiled. Only Simplifier.g and
     PolyPrinter.g might be affected.
     Call "ant" without parameters twice and you will
     notice that no compilation by Antlr is done.
 -->

<target name="PolyPrinter" depends="Simplifier">
    <antlr3 grammar.name="PolyPrinter.g"/>
</target>
<target name="Simplifier" depends="PolyDifferentiator">
    <antlr3 grammar.name="Simplifier.g"/>
</target>
<target name="PolyDifferentiator" depends="Poly">
    <antlr3 grammar.name="PolyDifferentiator.g"/>
</target>
<target name="Poly">
    <antlr3 grammar.name="Poly.g"/>
</target>

<target name="compile" depends="PolyPrinter" description="compile">
    <javac srcdir="."
           destdir="."
           deprecation="Yes"
           listfiles="Yes"
           includeantruntime="false">
        <compilerarg value="-Xlint:unchecked"/>
        <classpath>
            <pathelement path="${java.class.path}"/>
        </classpath>
    </javac>
</target>
```

```
</project>
```

The output generated should be similar to this:

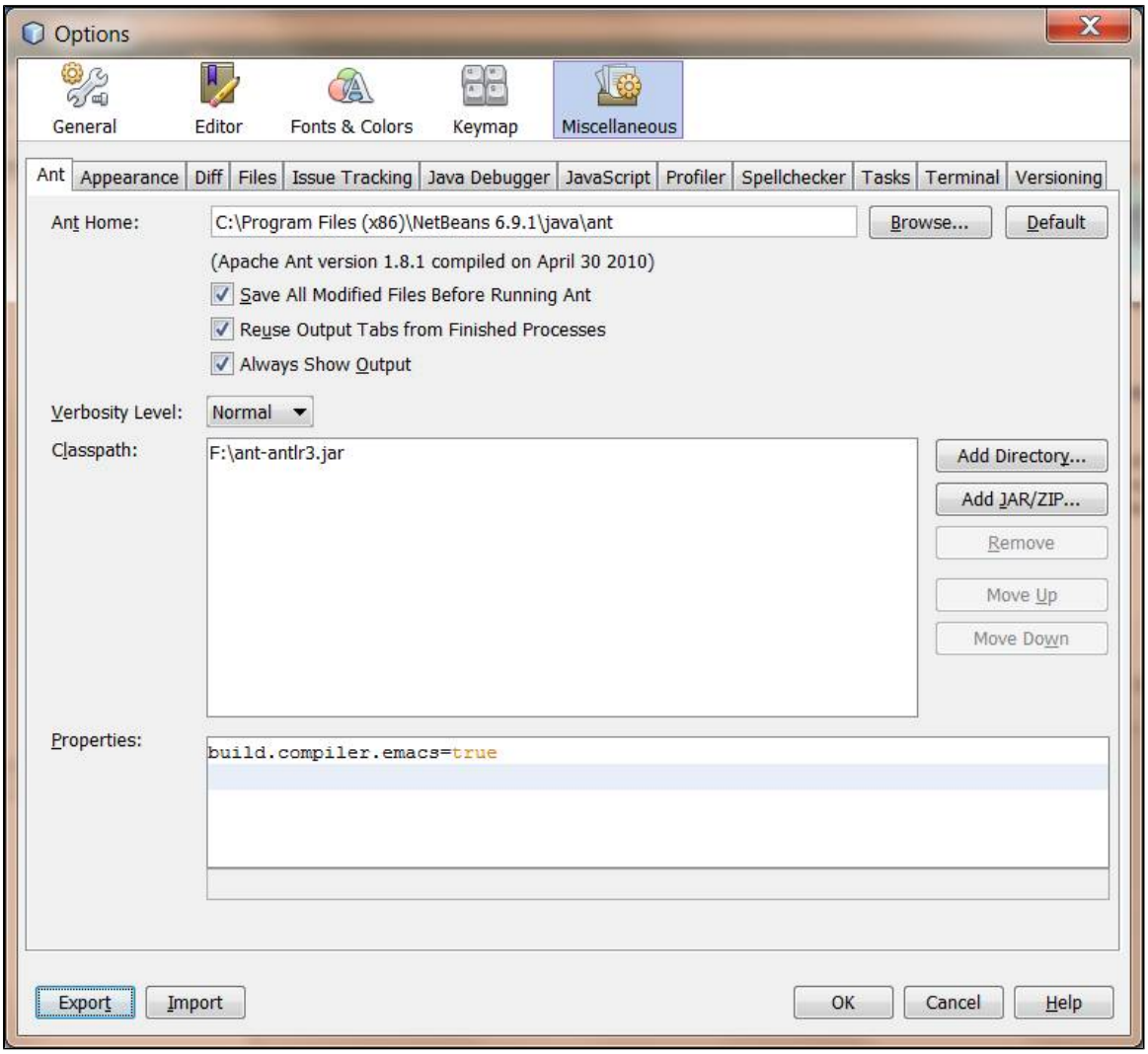
```
Buildfile: F:\examples-v3\Java\polydiff\build.xml

Poly:
[antlr:ant-antlr3] Failed to change file modification time
[antlr:ant-antlr3] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
[antlr:ant-antlr3] F:\examples-v3\Java\polydiff\Poly.g
[antlr:ant-antlr3] two-threaded DFA conversion
[antlr:ant-antlr3] Poly.poly:5:22 decision 1: k=1
[antlr:ant-antlr3] Poly.term:8:5 decision 2: k=3
[antlr:ant-antlr3] two-threaded DFA conversion

PolyDifferentiator:
[antlr:ant-antlr3] F:\examples-v3\Java\polydiff\PolyDifferentiator.g
[antlr:ant-antlr3] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
[antlr:ant-antlr3] warning(138): F:\examples-v3\Java\polydiff\PolyDifferentiator.g:0:0: grammar
PolyDifferentiator: no start rule (no rule can obviously be followed by EOF)
[antlr:ant-antlr3] two-threaded DFA conversion
[antlr:ant-antlr3] PolyDifferentiator.poly:9:5 decision 1: k=4

Simplifier:
[antlr:ant-antlr3] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
[antlr:ant-antlr3] F:\examples-v3\Java\polydiff\Simplifier.g
[antlr:ant-antlr3] two-threaded DFA conversion
[antlr:ant-antlr3] Simplifier.poly:15:5 decision 1: k=1 backtracks

PolyPrinter:
[antlr:ant-antlr3] F:\examples-v3\Java\polydiff\PolyPrinter.g
[antlr:ant-antlr3] ANTLR Parser Generator  Version 3.3 Nov 30, 2010 12:50:56
[antlr:ant-antlr3] warning(138): F:\examples-v3\Java\polydiff\PolyPrinter.g:0:0: grammar
PolyPrinter: no start rule (no rule can obviously be followed by EOF)
[antlr:ant-antlr3] two-threaded DFA conversion
[antlr:ant-antlr3] PolyPrinter.poly:8:5 decision 1: k=1

compile:
    [javac] Compiling 6 source files to F:\examples-v3\Java\polydiff
    [javac] F:\examples-v3\Java\polydiff\Main.java
    [javac] F:\examples-v3\Java\polydiff\PolyDifferentiator.java
    [javac] F:\examples-v3\Java\polydiff\PolyLexer.java
    [javac] F:\examples-v3\Java\polydiff\PolyParser.java
    [javac] F:\examples-v3\Java\polydiff\PolyPrinter.java
    [javac] F:\examples-v3\Java\polydiff\Simplifier.java
    [javac] F:\examples-v3\Java\polydiff\PolyPrinter.java:49: warning: [unchecked] unchecked call
to put(K,V) as a member of the raw type java.util.HashMap
    [javac]         super.put(attrName, value);
    [javac]                   ^
    [javac] F:\examples-v3\Java\polydiff\PolyPrinter.java:53: warning: [unchecked] unchecked call
to put(K,V) as a member of the raw type java.util.HashMap
    [javac]         super.put(attrName, new Integer(value));
    [javac]                   ^
    [javac] 2 warnings

BUILD SUCCESSFUL
Total time: 16 seconds
```

A second execution of the polydiff build without any changes of the grammar files will result in no new generation of java sources or a recompilation of the compile target.

```
Buildfile: F:\examples-v3\java\polydiff\buildantlr1.xml

Poly:

PolyDifferentiator:

Simplifier:

PolyPrinter:

compile:

BUILD SUCCESSFUL
Total time: 6 seconds
```

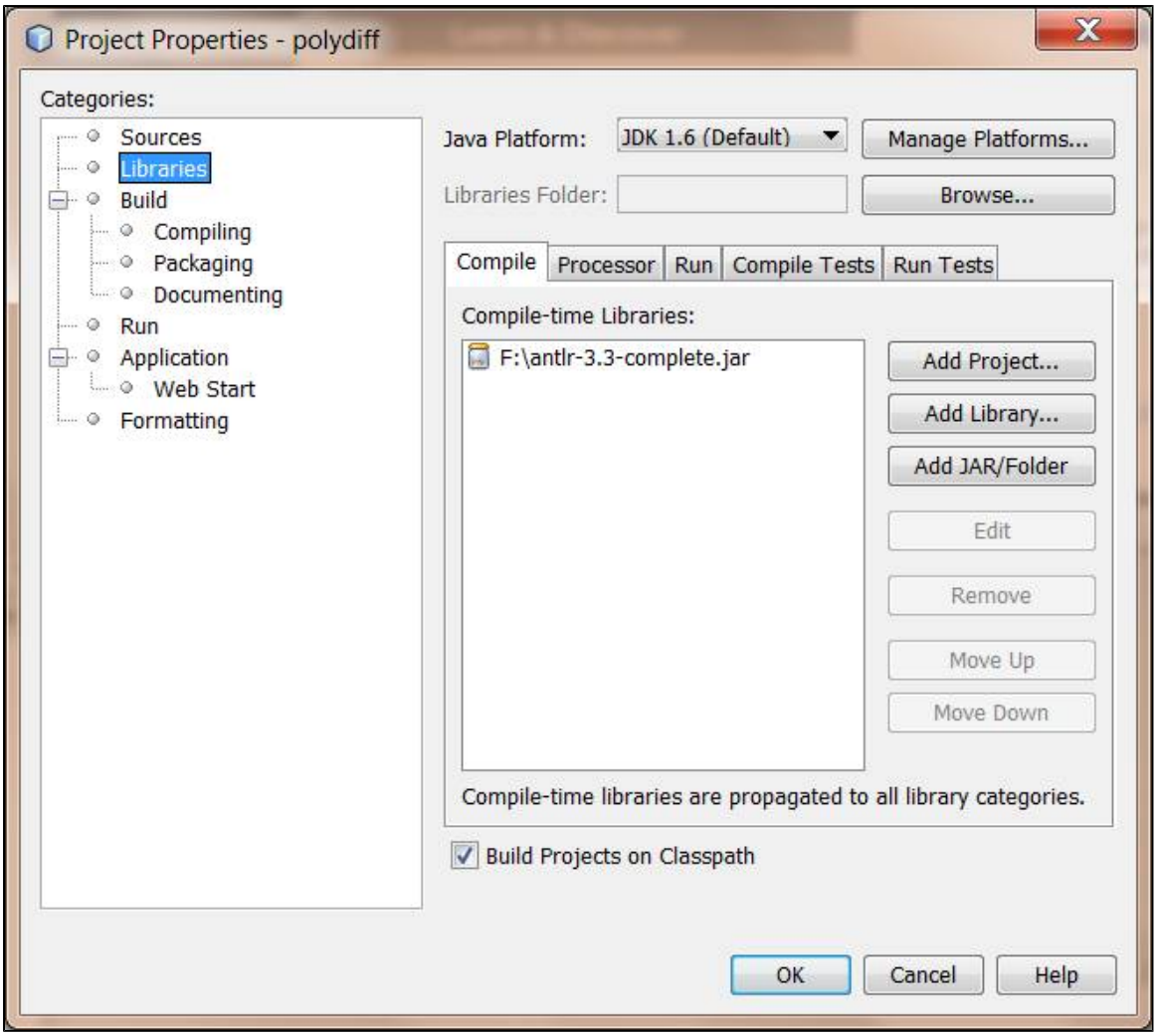
ANTLR3, the ANTLR3 task for ant and Netbeans

To give Netbeans access to the ANTLR3 task for ant, add the ant-antlr3.jar file to the ant classpath:



Equivalent is to copy the ant-antlr3.jar into the netbeans ant home/lib directory.

To make the ANTLR3 libraries available to the Netbeans classpath, use the project properties dialog:



Add the ANTLR3 libraries to the Compile- and the Run-Path.

The ANTLR3 libraries can be referenced in the netbeans build file via the ${javac.classpath} and the ${run.classpath} properties.

```
    ...
    <!-- where to place the antlr generated files -->
    <property name="src" location="src"/>
    <!-- name of the package -->
    <property name="package" value=""/>
    <!-- where to find the grammar files -->
    <property name="grammar" location="grammar"/>
    <!-- where to write/find token files -->
    <property name="token.lib" location="${src}/${package}" />

    <!-- where to place the antlr generated files -->
    <property name="src" location="src"/>
    <!-- name of the package -->
    <property name="package" value=""/>
    <!-- where to find the grammar files -->
    <property name="grammar" location="grammar"/>
    <!-- where to write/find token files -->
    <property name="token.lib" location="${src}/${package}" />

    <!-- a convenience macro which invokes antlr -->
    <macrodef name="antlr3">
        <attribute name="grammar.name"/>
        <attribute name="package" default="${package}"/>
        <attribute name="multithreaded" default="True"/>
        <attribute name="verbose" default="True"/>
        <attribute name="debug" default="False"/>
        <sequential>
            <echo message="antlr ${grammar}/@{grammar.name}" />
            <antlr:ant-antlr3 xmlns:antlr="antlib:org/apache/tools/ant/antlr"
                  target="${grammar}/@{grammar.name}"
                  outputdirectory="${src}/@{package}"
                  libdirectory="${src}/@{package}"
                  multithreaded="@{multithreaded}"
                  verbose="@{verbose}"
                  debug="@{debug}">
                <classpath>
                    <pathelement path="${javac.classpath}"/>
                </classpath>
            </antlr:ant-antlr3>
        </sequential>
    </macrodef>

    <!-- Modify here. Adapt the dependencies for your grammar(s).
         Watch out to specify the dependencies in the correct order!
         The sequence in which the grammars are compiled by Antlr in
         this example:
         1: Poly.g as PolyDifferentiator depends on it
         2: PolyDifferentiator as Simplifier dependends on it
         3: Simplifier.g as PolyPrinter dependends on it
         4: PolyPrinter.g
    -->
    <target name="-pre-compile" depends="PolyPrinter"/>
    <target name="PolyPrinter" depends="Simplifier">
        <antlr3 grammar.name="PolyPrinter.g"/>
    </target>
    <target name="Simplifier" depends="PolyDifferentiator">
        <antlr3 grammar.name="Simplifier.g"/>
    </target>
    <target name="PolyDifferentiator" depends="Poly">
        <antlr3 grammar.name="PolyDifferentiator.g"/>
    </target>
    <target name="Poly">
        <antlr3 grammar.name="Poly.g"/>
    </target>

</project>
```

Environment variables supported by ANTLR3 task for ant

The ANTLR3 task for ant supports two environment variables:

1. ANTLR_LIBDIR
   Should point to the directory which contains the ANTLR3 libraries

```
+- lib directory
   - antlr*.jar files
   - antlr-runtime-*.jar files
   - stringtemplate-*.jar files
```

   The ANLTR3 task for ant looks for all "antlr-*.jar", "stringtemplate-*.jar", "antlr-runtime-*.jar" files in the directory $ANTLR_LIBDIR points to and adds them to the classpath of the ANTLR3 Tool. As wildcards are being used for the version part of the jar-archives, it makes the task independent of new releases.

2. ANTLR_HOME
   ANTLR_HOME tries to find the ANTLR libraries in the $ANTLR_HOME/lib directory

```
+- antlr_home directory
   +- lib directory
      - antlr*.jar files
      - antlr-runtime-*.jar files
      - stringtemplate-*.jar files
```

   The ANLTR3 task for ant looks for all "antlr-**.jar", "stringtemplate-**.jar", "antlr-runtime-*.jar" files in the lib subdirectory $ANTLR_HOME points to and adds them to the classpath of the ANTLR3 Tool. As wildcards are being used for the version part of the jar-archives, it makes the task independent of new releases. In the event that no such jar-archives were found, a second search is done in the directory ANTLR_HOME points to.

Always use only one way to reference the ANTLR3 libraries:

1. Either let the ANTLR libraries be part of the CLASSPATH
2. or use the ANTLR_LIBDIR environment variable
3. or use the ANTLR_HOME environment variable

The next lines show how to make sure that the ANLTR3 libraries can be found via the ANTLR_LIBDIR environment variable.

```
<project name="anltr3" default="antlr_libdir" basedir=".">

    <!-- Inquire environment variables -->
    <property environment="envList"/>

    <!-- Get environment variable ANTLR_LIBDIR -->
    <property name="antlrLibDir" value="${envList.ANTLR_LIBDIR}"/>

    <!-- Check if ANTLR_LIBDIR environment variable is set -->
    <condition property="antlrLibDir.isset">
        <isset property="envList.ANTLR_LIBDIR"/>
    </condition>

    <target name="antlr_libdir">
        <fail message="ANTLR_LIBDIR environment variable not defined">
            <condition>
                <not>
                    <isset property="antlrLibDir.isset"/>
                </not>
            </condition>
        </fail>

        <whichresource property="antlr.jar.location" class="org.antlr.Tool"
classpathref="antlr.path"/>
        <fail message="Antlr library not found via ANTLR_LIBDIR='${antlrLibDir}'">
            <condition>
                <not>
                    <isset property="antlr.jar.location"/>
                </not>
            </condition>
        </fail>
        <echo>ANTLR3 found via environment variable ANTLR_LIBDIR: ${antlr.jar.location}</echo>
    </target>

    <!-- Use wildcards in pattern definition -->
    <!-- to be independent of antlr versions -->
    <patternset id="antlr.libs">
        <include name="antlr-*.jar" />
    </patternset>

    <!-- Looking for archives in ANTLR_LIBDIR -->
    <path id="antlr.path">
        <fileset dir="${antlrLibDir}" casesensitive="yes">
            <patternset refid="antlr.libs" />
        </fileset>
    </path>

</project>
```

As it resembles the previous examples very much, only a snippet of a build file using the ANLTR_LIBDIR environment variable is shown. The important difference is the reference to "antlr.path", as defined in the preceding build-file, instead of the ${java.class.path} in the antlr3-task for ant and the javac-task.

```
<project name="anltr3" default="compile" basedir=".">
    ...

    <!-- Antlr3 is called here -->
    <target name="antlr">
        <antlr:ant-antlr3 xmlns:antlr="antlib:org/apache/tools/ant/antlr"
                verbose="true"
                outputdirectory="."
                target="CMinus.g">
            <classpath>
                <path refid="antlr.path"/>
            </classpath>
            <jvmarg value="-Xmx512M"/>
    </target>

    <target name="compile" depends="antlr" description="compile">
        <javac srcdir="."
               destdir="."
               deprecation="Yes"
               listfiles="Yes"
               includeantruntime="false">
            <compilerarg value="-Xlint:unchecked"/>
            <classpath>
                <path refid="antlr.path"/>
            </classpath>
        </javac>
    </target>

</project>
```

ANTLR3 task for ant parameters

Attribute	Description	Required
Target	The grammar file to process.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
libdirectory	The directory where to find token files.	No
depend	When set to “true”, ANTLRs ‘depend’ option is used to resolve dependencies and to decide whether to invoke ANTLR for compilation. When set to “false”, try to figure out if an ANTLR generated file is out of date without invoking ANTLR with its ‘depend’ option. Default setting is “false” to keep backwards compatibility.	No
report	When set to "true", print out a report about the grammar(s) processed. Default is “false”.	No
print	When set to "true", print out the grammar without actions. Default is “false”.	No
debug	When set to "true", the generated parser emits debugging events. Default is “false”	No
profile	When set to "true", generates a parser that computes profiling information. Default is “false”.	No
trace	When set to "true", generates a parser that computes profiling information. Default is “false”.	No
nfa	When set to "true", generate an NFA for each rule. Default is “false”.	No
dfa	When set to "true", generate an DFA for each rule. Default is “false”.	No
messageFormat	When set to a message format the specified output style for messages is used. Default is “false”.	No
verbose	When set to "true", generates ANTLR version and other information	No

make	When set to "true", only build if generated files older than grammar. If set to True the depend option is ignored.	No
multithreaded	When set to "true", run the analysis in 2 threads. Default is “false”.	No
dir	The directory to invoke the VM in.	No
dbgST	When set to “true”, put tags at start/stop of all templates in output. Default is “false”.	No
noprune	Test lookahead against EBNF block exit branches. Default is “false”.	No
nocollapse	collapse incident edges into DFA states Default is “false”.	No
maxinlinedfastates	max DFA states before table used rather than inlining Default is 10 as per ANTLR3	No

When should I use the ANTLR3 task for ant ?

Here are some statements which should help you to decide if you are better off using the ANTLR3 task for ant or the Java task.

Reason	ANTLR3 task for ant	Ant Java-Task
Availability of new or modified ANTLR3 command-line options	Have to be implemented in the ANLTR3 task for ant to be available.	New or modified ANTLR3 command-line arguments can be used at once.
Access to all ANTLR3 command-line options	A subset of the most important command-line options is provided. In most cases this will be no restriction.	All command-line options can be accessed.
Build of grammar file only when required	Yes.	Yes. Has to be activated by -make option.
Seamless ant integration	Yes.	A little bit bumpy and a little bit limited.