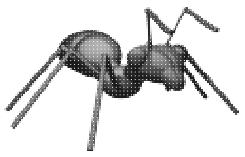
Ant DTD

Michel Casabianca
casa@sweetohm.net



Why a DTD for Ant ? Because good XML editors (such as Emacs with PSGML) can ease *build.xml* files writing only if a **DTD** can be parsed. For instance, Emacs + PSGML indicates possible elements for a point in a file, prompts for mandatory attributes (in the minibuffer), etc. So a DTD is nice.

Furthermore, a DTD is mandatory in some cases. For instance if you want to include a file A in a file B. Let's consider the following file :

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE project PUBLIC "-//ANT//DTD project//EN" "project.dtd" [
<!ENTITY include SYSTEM "message.xml">
]>

<project name="test" default="task" basedir=".">

 <target name="task">
  <echo message="Hello"/>
  &include;
 </target>

</project>
```

It imports, with an entity in internal subset, the file *message.xml* :

```
<echo message="World !"/>
```

Then Ant ouputs :

```
$ ant
Buildfile: build.xml
Project base dir set to: /home/casa/tmp/test
Executing Target: task
Hello
World !
Completed in 2 seconds
```

The task defined in the file *message.xml* have been imported in the *build.xml* file and processed by Ant.

I wrote such a DTD (it was painfull, believe me), you can download it here.

Maybe you wonder how it is possible to write such a DTD while users can define their own elements (with the <taskref> element).

To solve this problem, there are two ways :

Internal subset

The first solution is to define an element associated with a task in the internal subset of the DTD (within the DOCTYPE element).

For instance, if you declare a task as :

```
<xt xml="file.xml" xsl="file.xsl" out="file.html"/>
```

You may add to DOCTYPE the following fragment :

```
<!ENTITY % ext "| xt">

<!ELEMENT xt EMPTY>
<!ATTLIST xt
          xml CDATA #REQUIRED
          xsl CDATA #REQUIRED
          out CDATA #REQUIRED>
```

The role of <!ELEMENT> and <!ATTLIST> elements are obvious. But the entity <!ENTITY % ext "| xt"> may seem strange. This entity is appended to the content definition of the element `target` in the DTD :

```
<!ELEMENT target (ant | ... | zip %ext;)*>
```

The fragment `| xt` is appended to the content model so you can include an <xt> element in <target>.

Extensions file

If you want an extension to be declared in all files, you add it to the *project-ext.dtd* file :

```
<!ENTITY % ext "| xt">

<!ELEMENT xt EMPTY>
```

```
<!ATTLIST xt
          xml CDATA #REQUIRED
          xsl CDATA #REQUIRED
          out CDATA #REQUIRED>
```

The idea is the same, but the fragment is not declared in the DTD subset, instead it is in the extension file that is included in the DTD with :

```
<!ENTITY % ext-file SYSTEM "project-ext.dtd">
%ext-file;
```

The method you choose depends on the accessibility you would desire to this extension. This is very similar to placing extensions in a separate jar file or in Ant's jar. In the latter case, extensions are allways reachable, whatever *build.xml* file you work on.