# DTD Ant

*Michel Casabianca*
*casa@sweetohm.net*

Vous avez besoin d'une DTD pour vos fichiers de build Ant, et voici pourquoi !

Pourquoi une DTD pour Ant ? Parcequ'un bon éditeur XML (comme Emacs associé à PSGML) permet de faciliter la tâche lors de l'écriture des fichiers *build.xml* si et seulement si l'on dispose d'une **DTD** pour ces fichiers. Par exemple, le couple Emacs + PSGML indique les éléments autorisés en un point du fichier, affiche une invite (dans le minibuffer) pour compléter les attributs obligatoires, etc. Bref, une DTD facilite la vie.

De plus, une DTD est obligatoire dans certains cas. Par exemple lorsqu'on souhaite inclure un fichier A dans un fichier B. Considérons le fichier suivant :

```
<?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>
```

Il importe, au moyen d'une entité dans la sous DTD interne, le fichier *message.xml* que voici :

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

Le résultat de l'appel de Ant est alors :

```
$ ant
Buildfile: build.xml
Project base dir set to: /home/casa/tmp/test
Executing Target: task
```

```
Hello
World !
Completed in 2 seconds
```

On constate que la tâche définie dans le fichier *message.xml* a bien été importée dans le fichier *build.xml* et exécutée par Ant.

Je me suis attelé à la tâche (longue et pénible) d'écrire cette DTD, que l'on peut télécharger ici.

On peut cependant se demander comment il est possible de définir une telle DTD alors que le jeu des éléments autorisés n'est pas fixé (puisqu'il est possible de définir ses propres tâches grâce à l'élément <taskdef>).

Pour contourner le problème, il existe deux solutions :

# Sous DTD interne

La première solution consiste à définir les éléments associés aux tâches définies par l'utiisateur à l'aide de la sous DTD interne (le fragment de DTD se trouvant dans l'élément DOCTYPE).

Par exemple, si on définit une tâche ayant la forme suivante :

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

On pourra ajouter dans le DOCTYPE le fragment suivant :

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

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

Les rôles de <!ELEMENT> et de <!ATTLIST> sont évidents, par contre, l'entité <!ENTITY % ext "| xt"> peut paraître plus mystérieuse. Cette entité est collé à la définition de l'élément `target` dans le fichier de la DTD, comme suit :

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

Le fragment `| xt` est donc ajouté à la définition et permet alors d'intégrer des éléments <xt> dans un élément <target>.

# Fichier d'extensions

Pour qu'une extension soit disponible dans tous les fichiers *build.xml*, on peut l'ajouter dans le fichier *project-ext.dtd* qui a l'allure suivante :

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

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

Le principe est le même que dans la méthode précédente, mais ce fichier est inclus dans la DTD des projets de manière permanente, à l'aide de :

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

Le choix de la méthode de définition des extensions dépendra de la disponibiité souhaitée pour l'extension (de la même manière que l'on peut inclure les classes Java d'une extension dans un jar séparé ou dans celui des classes de Ant, pour qu'elles soient toujours disponibles).