# Laboratoire

## L'UTILITAIRE «ANT»

**OBJECTIF** Utiliser et entrevoir les possibilités de l'utilitaire «**ant**»

**BIBLIOGRAPHIE** Documentation mise à disposition par l'outil lui-même

# *Contenu*

# 1 Préambule

«**Ant**» est un outils de construction, à l'image de «**make**» du système Unix.

Si on développe un applicatif Java, cet outil permet notamment de spécifier la suite des commandes qui permettront par exemple de compiler le programme en présisant le classpath, d'indiquer le dossier qui contiendra le résultat de la compilation, de contruire le jar exécutable, de créer des dossiers intermédiaires, de nettoyer les mêmes dossiers une fois la construction effectuée, etc..

Les environnements commerciaux à succès tels que NetBeans et Eclipse s'appuient sur **ant** pour mettre en oeuvre les commandes de compilation et d'exécution. La connaissance de l'outil **ant** peut permettre le cas échéant de personnaliser l'utilisation de NetBeans ou d'Eclipse.

Si l'on désire au contraire faire "simple" et efficace, développer son applicatif avec un simple éditeur de texte, **ant** peut être un complément idéal, à même d'effectuer les opérations de compilation et d'exécution.

Ant est un outil écrit en Java.

# 2 Installation

- Télécharger «Ant» depuis le site http://www.apache.org/ant/ (Octobre 2009 > version `apache-ant-1.7.1-bin.zip`)
- Au moyen de Unzip, placer les fichiers dans `c:\javatools\ant` (par exemple)
- Pour Windows: ajouter - ou mettre à jour - les variables d'environnement:
  - **`ANT_HOME`** = `c:\javatools\ant`
  - `c:\javatools\ant`**`\bin`** à rajouter comme chemin dans la variable système `Path`
  - **`JAVA_HOME`** = `C:\Program Files\Java\jdk1.6.0_10` (répertoire d'installation de votre Java Development Kit).
- Pour Mac: il suffit de passer directement au point suivant.
- Pour tester l'installation, taper «`ant`» dans une fenêtre de commande, le message «`Buildfile: buid.xml does not exist` !» devrait s'afficher.

**Si ça ne fonctionne pas..**

Contrôlez les points suivants:

- Contrôler que vous avez téléchargé la bonne version de **ant**, celle qui contient notamment un répertoire `bin`.
- Contrôler que l'environnement ant a été décompressé directement sous `c:\javatools\ant`, sans avoir rajouté de répertoire intermédiaire comme par exemple `c:\javatools\ant\`**`ant1.7.1`**`\..`

**Sous Windows:**

- Existance d'une variable d'environnement **`JAVA_HOME`** (définie au niveau du système d'exploitation), indiquant le chemin du répertoire d'installation du JDK.
- Supprimer toute variable d'environnement **`CLASSPATH`** qui aurait été définie au niveau du système d'exploitation.
- Contrôler que le JDK n'est pas installé dans le répertoire «`Program Files`». De manière générale, il est recommandé avec Java d'éviter les noms de répertoires qui contiennent des espaces.
- Si ça ne fonctionne toujours pas, prenez l'initiative de purger votre disque dur et de revoir toutes vos installations.
- Si ça ne fonctionne toujours pas, demandez l'aide de l'assistant.

# 3 Un premier exemple

## Le fichier `build.xml`

«`build.xml`» est le fichier qui sera utilisé par l'utilitaire ant dans le but de construire et de déployer notre application.

Voici le code d'un petit fichier **build.xml** qui compile un programme source Java situé dans le répertoire **src** et place les fichiers résultant de la compilation dans le répertoire **classes**.

```
<?xml version="1.0"?>
<!-- Un petit fichier de construction -->
<project name="Ant premier test" default="build" basedir=".">
     <target name="build" >
          <javac srcdir="src" destdir="classes" debug="true"
          includes="**/*.java"
          />
     </target>
</project>
```

- La première ligne du fichier `build.xml` représente la déclaration du type de document (un fichier xml)
- La ligne suivante est un commentaire.
- La troisième ligne constitue la balise «`project`». Chaque fichier de construction contient toujours une balise «`project`», et une seule. Toutes les instructions sont placées à l'intérieur de cette balise.

## La balise «`project`»

```
<project name="Ant premier test" default="build" basedir=".">
```

Un fichier `build.xml` ne peut contenir qu'une seule balise project, qui est par contre obligatoire ! Cette balise requiert 3 attributs qui seront expliqués d'ici quelques lignes..

## La balise «`target`»

Un même projet peut contenir un ou plusieurs «**targets**»: un «target» est une **tâche** qui regroupe un certain nombre **d'actions** à accomplir. Notre exemple est pour l'instant limité à un seul «target»: «`build`», qui utilise la commande **`javac`** pour compiler les fichiers java.

```
<target name="build" >
          <javac srcdir="src" destdir="classes" debug="true"
          includes="**/*.java"
          />
</target>
```

Si un même projet contient plusieurs «targets», ces derniers ne seront pas tous exécutés pour autant ! C'est en saisissant la commande **`ant`** que l'on peut choisir la tâche que l'on désire exécuter:

| | |
|---|---|
| `> ant x` | Exécution du target «x» |
| `> ant y` | Exécution du target «y» |
| `> ant` | Exécution du target spécifié par défaut *(voir ci-dessous l'attribut `default` de la balise `project`)* |

### Les attributs de la balise `project`

La balise project requiert 3 attributs **`name`**, **`default`** et **`basedi`**r, à spécifier obligatoirement !

| Attribut | Description |
|---|---|
| `name` | Nom du projet. |
| `default` | Nom du target principal par défaut (*default target*) à utiliser lorsqu'aucun target n'est spécifié en ligne de commande.:<br><br>> **`ant`** *<return>* -- *Utilisation du target par défaut*<br><br>> **`ant`** *nomTarget* -- *Utilisation du target «nomTarget»* |
| `basedir` | Nom du répertoire de base à partir duquel toutes les constructions de chemins seront opérées. |

**EXERCICE 1**

☞ Tester le fichier build.xml avec un petit programme Java "Hello the world".

# 4 En savoir plus sur Ant ?

Consulter la documentation de ant accessible à partir de votre répertoire d'installation:
`C:\rep_installation\docs\manual\index.html`

Allez succcessivement dans:

```
> Ant Tasks

    > Core Tasks

        > javac
```

Vous y trouvez la description de la tâche de compilation **`javac`**, telle qu'elle a été utilisée dans l'exemple de l'exercice 1.

Sinon, dans la documentation générale, il est dit qu'un *target* peut dépendre de plusieurs targets. On peut par exemple avoir un target pour compiler, et un target pour déployer l'exécutable. Or, on peut déployer l'exécutable uniquement si ce dernier a d'abord été compilé.. Ainsi, le target de déploiement est donc **dépendant** du target de compilation.

Ant essaye d'exécuter les targets indiqués dans l'attribut **depends** dans l'ordre dans lequel ils apparaissent (de gauche à droite). Il est possible qu'un target soit déjà exécuté si par exemple un target précédent dépendait déjà de ce dernier.

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Supposons que l'on désire exécuter le target D. On pourrait supposer, en observant son attribut de dépendance, que les targets seront exécutés en suivant l'ordre C, puis B, puis A. Il n'en est rien! C dépend de B, et B dépend de A. Ainsi, A sera d'abord exécuté, puis B, puis C, et finalement D.

**EXERCICE 2**

☞ Consulter la documentation ant et rechercher le mode d'emploi de la tâche d'exécution «`java`». Rajouter le target «**`exec`**» dans le fichier «`build.xml`» en spécifiant qu'il s'agit du target par défaut et en spécifiant ce dernier avant le target «build». La commande `java` utilisera notamment l'attribut **`classpath`** pour spécifier le chemin d'accès aux différentes classes (`classes/`).

**EXERCICE 3**

# ☞ Créer son propre environnement de développement

Vous allez créer un modèle de répertoire dédié au développement d'une petite application Java.

*!! Ce modèle pourra par la suite être conservé, utilisé et adapté dans le cadre de vos futurs développments !!*

Le répertoire «modèle» contiendra les éléments décrits ci-après:

| Répertoire | Description |
|---|---|
| `monApplicJava` | Répertoire de base, contenant:<br>❍ le fichier **build.xml**, utilisé par Ant,<br>❍ le répertoire **`src`** (voir ci-dessous)<br>❍ le répertoire **`lib`** (voir ci-dessous) |
| **`/src`** | Pour placer tous les fichiers source **`.java`** de l'application à développer. |

| | |
|---|---|
| **/lib** | Pour placer tous les paquetages annexes requis par l'application. Ces paquetages seront stockés sous la forme de fichiers ***.class** (déjà compilés) |

**A/** Mettez en place l'architecture suivante:

**monApplicJava**
monApplicJava/**src**
monApplicJava/**lib**

A titre d'illustration, nous allons travailler avec une application Java réalisant une annonce tournante, constituée:

1. Du fichier **Gui.java,** qui vous est donné, à placer dans monApplicJava/**src**
2. Du répertoire **utilitaires**, un paquetage contenant les deux fichiers **Fenetre.class** et **Fenetre$FenetreEcouteur.class**, qui vous sont donnés déjà compilés. A placer dans monApplicJava/**lib.**

**B/** Assurez-vous dans une première étape que vous êtes capables de compiler, d'exécuter cette application, de créer un jar exécutable sans faire appel à l'outil **ant**.

**Rappel**: comment créer un jar exécutable?

*Voici la commande:*

```
jar cvmf MANIFEST.MF Programme.jar *.class rep1 rep2
```

Explication : jar est tout simplement la commande utilisée par Java pour gérer les fichiers JAR.

La suite de lettres "cvmf" :

• 'c' : veut dire que vous voulez créer un fichier JAR (il faut en effet le préciser car la commande jar peut aussi être utilisée pour ouvrir un fichier .jar par exemple).

• 'v' : indique que vous voulez afficher les détails sur la création du JAR (mode "verbeux").

• 'm' : indique que vous voulez utiliser votre propre fichier MANIFEST.MF (ce qui n'est pas forcément utile dans le cas d'une archive non-exécutable).

• 'f' : indique que le résultat sera un fichier (si vous omettez de l'indiquer, le résultat s'affichera dans la console ce qui ne servirait à rien).

La suite indique le nom du fichier MANIFEST (ici MANIFEST.MF), le nom que vous voulez donner au fichier JAR qui sera créé (ici Programme.jar)

Enfin les fichiers que vous voulez inclure dans l'archive. Les séparer par des espaces. S'il s'agit simplement d'un nom de répertoire, ce dernier sera intégré dans l'archive de manière récursive, avec tous ses sous-répertoires.

Le fichier **manifest.fm** ?

Ce fichier est en fait un simple fichier texte contenant diverses informations comme :

• le nom de la classe principale (contenant la méthode main) ;

• le CLASSPATH (chemin) menant à d'éventuelles archives..

Son format ! DOIT SE TERMINER PAR UN RETOUR DE LIGNE !

```
Main-Class: ClassePrincipale
Class-Path: .\MesJars\x.jar
```

Dans notre exemple, seule la spécification du main-class suffira.

**C/** Créer le fichier **`build.xml`** qui aura pour tâche:

1. de créer les deux répertoires **`/classes`** et **`/temp`**
2. de compiler l'application (javac) en plaçant le résultat de la compilation dans **`/classes`**.
3. de construire le jar exécutable, quisera placé directement dans le répertoire **`monApplicJava,`** en copiant les fichiers résultant de la compilation dans le répertoire **`/temp`**, et en copiant (toujours dans **`/temp`**) tout le contenu de **`/lib`** (paquetages requis). Le contenu du jar exécutable contiendra une compression de tous ces fichiers.
4. de nettoyer les répertoires..Effacer le répertoire "`classes`" ainsi que le répertoire "`temp`"
5. d'exécuter le jar créé dans le point précédent.

 **Voici le début du fichier build.xml, à compléter par vos soins, - le demander à l'assistant(e) - Vous noterez que le target «`init`» définit des «*properties*». Ces variables peuvent être utilisées par la suite en utilisant la syntaxe `${`*`nomVariable`*`}`**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- ..................................................  -->
<!-- Fichier de construction standard                    -->
<!-- build.xml, Octobre 2009                        -->
<!-- Eric Lefrançois                                     -->
<!-- ..................................................  -->


<project name="Standard" default="all" basedir=".">

    <target name="init">

        <!-- Nom logique des répertoires -->
        <property name="root" value="${basedir}"/>
        <property name="libDir" value="${root}/lib"/>
        <property name="srcDir" value="${root}/src"/>
        <property name="classesDir" value="${root}/classes"/>
        <property name="tempDir" value="${root}/temp"/>

        <!-- Autres -->
```

*Compléter ici par la définition de vos prores "properties", de manière à écrire vos targets de la manière la plus génériqoue qui soit.*

```
        <!-- Création des répertoires de classe et de construction -->
```

*Compléter..*

```
      <!-- Liste des targets -->
```

*Compléter.. Ecriture de vos "targets"*

```
    <!-- Target principal -->
    <target name="all" depends="execution"/>

</project>
```