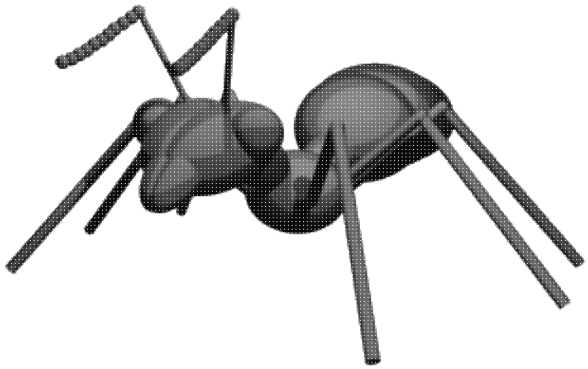
Sweet Ant Tools

Michel CASABIANCA
casa@sweetohm.net

SAT est un ensemble d'outils pour Ant sous licence Apache. On y trouve en particulier un mode console (pour lancer une console Ant où l'on peut taper des commandes pour lancer des builds par exemple) qui accélère considérablement la vitesse d'exécution. On y trouve aussi nombre de tâches, pour exécuter des scripts Beanshell depuis un script Ant, une tâche de transformation XSLT utilisant XT et d'autres destinées à manipuler des documents XML (les fusionner, les éclater ou encore les inclure dans un autre fichier). Un mode Ant pour Emacs est aussi fourni. Ce package est utilisé pour générer mon site web ainsi que celui des Éditions O'Reilly.



Installation

Archive Zip sat-1.0.zip.

Ce programme demande une machine virtuelle Java récente (1.2 ou postérieure). Pour l'installer, dézipper l'archive et recopier les fichiers du répertoire *lib* dans celui de Ant.

Compiler SAT

Pour compiler SAT, vous devez installer Ant version 1.3 ou 1.4. Vous pouvez le télécharger sur le site d'Apache. On devra aussi installer les jars de Ant (*ant.jar*) et d'un parser JAXP (par exemple, celui que l'on peut télécharger sur le site de Sun) dans le répertoire *lib* de SAT.

Pour générer le fichier jar contenant les classes de SAT, se placer à la racine de votre installation de SAT taper `ant jar` sur la ligne de commande. Placer ensuite le fichier jar généré (*sat.jar* du répertoire *lib*) dans le répertoire *lib* de Ant. On pourra alors générer les documentations et l'archive zip en tapant `ant` (SAT s'utilise lui même pour compiler les documentations et l'archive zip).

Console Ant

Cet outil permet de lancer Ant en mode console (dans un terminal ou la console d'un IDE par exemple). On y tape des commande qui sont interprêtées par la console pour lancer les targets du build file chargé. L'intérêt de cette console est d'accélérer grandement la vitesse d'exécution car on économise le temps de lancement de la machine virtuelle, de compilation par le JIT, de chargement du projet, etc. On peut ainsi espérer des gains d'un facteur 5. Il va sans dire que le gain est d'autant plus important que le temps d'exécution de ma target est court.

Installation

Aller dans le répertoire *bin* de votre installation de Ant. Recopier le fichier *ant* (ou *ant.bat* pour les handicapés de l'OS) sous le nom *antc* (ou *antc.bat*). Remplacer, dans ce nouveau fichier, toutes les occurences de `org.apache.tools.ant.Main` par `net.cafebabe.sat.ant.Console`. Ce script lancera alors la console Ant en lieu et place de Ant lui même. S'assurer que ce ficher est dans votre `PATH`.

Il faut bien entendu installer SAT. Pour ce faire, recopier le fichier *sat.jar* dans la répertoire *lib* de votre installation de Ant.

Utilisation

Pour lancer la console Ant, se placer dans le répertoire de son projet (où se trouve votre fichier de projet *build.xml*) et taper `antc`. On obtient une aide sur les paramètres en ligne de commande en tapant `antc -help` :

```
Ant Console 0.9 (C) Michel Casabianca 2003
type "help" to get help on console commands
Usage: antc [-help] [-version] [-emacs] [-timer] [-file file] [-find file]
-help    Print this help screen
-version Print the version
-timer   Print build times
-file    To set the build file
-find    To search for the build file
```

Lorsqu'on lance la console Ant, on obtient une invite du type :

Console Ant en action

Pour obtenir de l'aide sur les commandes que l'on peut lancer dans la console, taper `help` à l'invite. On obtient la page d'aide suivante :

```
Commands you can run in the console are the following:
  help          To display this help screen
  exit          To quit the console
  desc          To describe the loaded project
  load file     To load the build file
  find file     To find the project file
  reload        To reload the current project
  timer on/off  To set timer on/off
  target foo    To run the target foo
  targetname    To run the target (can't be a console command)
  <empty>       To repeat the last command
```

On notera que pour lancer l'exécution d'un target, il suffit de taper son nom à l'invite (c'est un raccourci pour la commande `target` si le nom de la target n'est pas celui d'une commande de la console). On notera aussi que l'on peut répéter la dernière commande en tapant simplement `ENTER`.

Il possible de charger un autre build file avec la commande `load` (le chemin du fichier est alors relatif au répertoire courant) et de rechercher récusivement le build file dans le système de fichier avec la commande `find` (comparable au paramètre `-find` sur la ligne de commande de Ant). Il est aussi possible de recharger le projet courant avec la commande `reload`. C'est nécessaire lorsque le build file a changé sur le disque ou lorsqu'un projet a un comportement étrange (ce qui arrive parfois à la suite d'une erreur de build, certaines tâches supportant mal d'être relancées après une erreur).

La commande `desc` affiche une description du projet en cours (comme avec le paramètre `-projecthelp` de Ant).

Enfin, on peut afficher les temps d'exécution avec la commande `timer` ayant pour paramètre *on* ou *off*. C'est utile pour se convaincre que la console est bien plus rapide (c'est pour cela que j'affiche le résultat en millisecondes :o)

Notes

Dans la mesure où l'on économise le temps d'initialisation de la VM, de la compilation par le JIT et le temps de chargement du projet, il est clair que l'on obtient de meilleurs résultats pour des builds courts. Par exemple, l'exécution d'une tache qui ne fait rien, la console met une trentaine de millisecondes contre près de 5 secondes dans Ant (un gain de perfs d'un facteur 100 environ). Cependant le gain reste appréciable pour de vrais builds. Par exemple, la génération du projet SAT prend environ 12 secondes dans la console contre 25 secondes anviron avec Ant.

Ma tache `server` permet d'obtenir les memes résultats, mais sa mise en oeuvre est plus complexe et je ne l'intègre plus dans les dernières versions de SAT (on pourra encore la trouver dans la version 0.7).

La console a été développée avec la dernière version stable de Ant, soit la *1.5.3-1*. Faites moi savoir si vous avez testé avec une autre version.

Description des tâches

SAT est un ensemble de tâches optionnelles pour Ant, pour chaque tâche est donc associé un élément XML et une déclaration `taskdef` (pour indiquer quelle classe Java définit la tâche).

File, dir et fileset

Toutes ces tâches manipulent des fichiers et utilisent les attributs `file` et `dir` ainsi que l'élément encapsulé `fileset` pour la sélection des fichiers à traiter. Tous ces attributs ou élément ont la même syntaxe que voici:

- `file`: C'est un attribut contenant la liste (dont le séparateur est la virgule) des fichiers à traiter. Par exemple, `file="foo"` sélectionne le fichier *foo* et *file="foo,bar"* les fichiers *foo* et *bar*.
- `dir`: C'est un attribut contenant la liste (dont le séparateur est une virgule) des répertoires à traiter. Sélectionne tous les fichiers du répertoire, mais pas ceux d'un éventuel sous répertoire. Par exemple, `dir="foo"` sélectionne tous les fichiers du répertoire *foo*, quelque soit leur type. Les fichiers d'un répertoire *foo/CVS* ne seraient pas sélectionnés. `dir="foo,bar"` sélectionne quant à lui tous les fichiers des répertoires *foo* et *bar*.
- `fileset`: C'est un élément encapsulé permettant une sélection plus fine des fichiers. Voire la documentation de Ant pour plus de détails.

Les fichiers ou répertoires dont les chemins sont relatifs le sont par rapport au script Ant. Donc un attribut `file="foo"` dans le script */home/casa/build.xml* désigne le fichier */home/casa/foo*.

Il est possible, dans une même tâche, d'utiliser toute combinaison d'attributs `file` et `dir` et de l'élément `fileset`. Par exemple, l'élément:

```
<foo file="file" dir="dir">
  <fileset dir="dir2" includes="*.xml"/>
</foo>
```

Sélectionne le fichier *file*, les fichiers du répertoire *dir* et les fichiers XML du répertoire *dir2*.

Bsh

Description

Cette tâche exécute un script Beanshell. Ce script peut être contenu dans un fichier ou encapsulé dans l'élément de la tâche. On peut aussi passer des arguments au script avec l'attribut `args` ou une expression dans un script encapsulé.

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="bsh" classname="net.cafebabe.sat.bsh.BshTask"/>
```

La tâche Beanshell a été testée avec succès avec la version 1.1a18 (la version dont le jar est distribuée dans ce package). Elle ne fonctionne pas avec les versions 1.1a12 ou antérieures (l'interface `bsh.ConsoleInterface` était différente avant la version 1.1a16) et avec la version 1.2b1 (pour cause de bug).

Paramètres

Attribut	Description	Requis
file	Script(s) Beanshell à exécuter	Non
dir	Répertoire(s) des scripts à traiter	Non
args	Liste d'arguments (séparés par des virgules) à passer au script Beanshell. On peut accéder au tableau des arguments avec `this.interpreter.get("bsh.args");`	Non
reset	Indique si l'interpréteur Beanshell doit être réinitialisé avant l'exécution d'un nouveau script. Peut prendre les valeurs *true* ou *false*.	Non (*true* par défaut)

Éléments encapsulés

On peut sélectionner les script à exécuter à l'aide d'un élément `<fileset>` encapsulé. Voire la documentation de Ant pour de plus amples détails sur l'élément `fileset`.

Contenu textuel

Un script peut être écrit directement dans un élément `<bsh>`.

Propriétés Ant

Un script Beanshell peut accéder aux propriétés Ant à l'aide des commandes `setAntProperty()` et `getAntProperty()`. Voir plus bas pour un exemple.

Commandes Beanshell

Ce package propose des commandes Beanshell permettant d'accéder aux propriétés Ant (définies à l'aide d'éléments `property`) à l'intérieur d'un script Beanshell. Ces commandes ont la signature suivante:

- `void setAntProperty(String name,String value)`: donne la valeur *value* à la propriété nommée `name`. La propriété n'a pas à avoir été déclarée par un élément `property`.
- `String getAntProperty(String name)`: renvoie la valeur de la propriété Ant appelée *name*.

Une autre commande permet de résoudre un fichier relativement au script Ant en cours d'exécution. En effet, un chemin relatif dans un script Ant doit être interprété par rapport au répertoire du script (et non par rapport au répertoire courant). La signature de cette commande est la suivante:

- `File resolveAntFile(String file)`: retourne le fichier par rapport au répertoire du script Ant.

On notera que cette tâche définit une variable `antProject` dans l'interpréteur Beanshell. Cette variable contient une référence vers le projet Ant en cours d'exécution.

Exemples

Pour exécuter un script *bsh.bsh* du répertoire *test*, on écrira l'élément suivant:

```
<bsh file="test/bsh.bsh"/>
```

On peut aussi écrire le script directement dans l'élément `<bsh>`, comme suit:

```
<bsh>
  print("Hello World !");
</bsh>
```

On notera qu'un script encapsulé dans un élément `bsh` est parsé lors de l'exécution du script Ant. Il ne doit donc pas contenir les caractères < et & ou bien il doit être contenu dans un marqueur `CDATA`, comme suit:

```
<bsh>
  <![CDATA[
  print("<date>"+new Date()+"</date>");
  ]]>
</bsh>
```

Pour passer les arguments `foo` et `bar` au script, (comme s'ils lui étaient passés par la ligne de commande) on utilisera un attribut `args`:

```
<bsh file="test/bsh.bsh" args="foo,bar"/>
```

Pour finir, un exemple plus complexe:

```
<bsh file="test/bsh.bsh" args="foo,bar">
  nested="nested";
</bsh>
```

Où le script *test/bsh.bsh* est le suivant:

```
#!/usr/local/bin/bsh
// display arguments on the command line
args=this.interpreter.get("bsh.args");;
if(args!=null) {
```

```
    for(int i=0;i<args.length;i++)
        print("Argument "+i+": "+args[i]);
}
// display argument in nested script
print("Argument nested: "+nested);
```

Cet exemple produit l'affichage suivant:

```
bsh:
    [bsh] Arguments: foo bar
    [bsh] Executing nested script...
    [bsh] Executing script 'test/bsh.bsh'...
    [bsh] Argument 0: foo
    [bsh] Argument 1: bar
    [bsh] Argument nested: nested
```

Cette trace indique l'ordre d'évaluation du code beanshell:

1. Les arguments sont déclarés à l'interpréteur.
2. Puis le script encapsulé (s'il y en a un) est exécuté.
3. Pour finir, le ou les scripts des fichiers sélectionnés sont exécutés.

Le script suivant affiche la valeur de la propriété `foo` et donne une valeur à `bar`:

```
print(getAntProperty("foo");
setProperty("bar","Hello World!");
```

XTask

Description

Cette tâche utilise le processeur XT de James Clark pour effectuer une transformation XSLT d'un document XML. Ce processeur (dont le développement a été arrêté par l'auteur) est tenu pour le plus rapide des processeurs XSLT implémentés en Java.

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="xtask" classname="net.cafebabe.sat.xml.XTask"/>
```

Paramètres

Attribut	Description	Requis
file	Fichier(s) XML à transformer.	Non
dir	Répertoire(s) des fichiers XML à transformer.	Non

style	Indique la feuille de style à utiliser pour la transformation.	Oui
tofile	Donne le nom du fichier généré (s'il n'y en a qu'un seul).	Non
todir	Donne le répertoire des fichiers générés (s'il y en a plusieurs).	Non
extension	Extension des fichiers générés.	Non (*.html* par défaut)
force	Force la transformation même si le fichier généré existe et est plus récent que le fichier source XML et la feuille de style XSLT. Peut prendre les valeurs *true* ou *false*.	Non (*false* par défaut)

Éléments encapsulés

On peut sélectionner le ou les fichiers XML à transformer à l'aide d'un élément `<fileset>`. Voire la documentation de Ant pour plus de détails sur cet élément.

On peut de plus passer des paramètres au processeur XSLT à l'aide d'éléments `<arg>`. Par exemple, pour passer la valeur *bar* au paramètre `foo`, on inclura dans l'élément `<xtask>` l'élément:

```
<arg name="foo" value="bar"/>
```

L'élément `<xtask>` ne prend pas en compte de contenu textuel.

Exemples

Pour transformer le fichier *foo.xml* en *bar.html* à l'aide de la feuille de style *transfo.xsl*, on écrira l'élément suivant:

```
<xtask file="foo.xml"
       style="transfo.xsl"
       tofile="bar.html"/>
```

Pour passer la valeur *bar* au paramètre `foo`, on invoquera la transformation par:

```
<xtask file="foo.xml"
       style="transfo.xsl"
       tofile="bar.html">
  <arg name="foo" value="bar"/>
</xtask>
```

Pour transformer les fichiers XML du répertoire *xml* en HTML avec la feuille de style *page.xsl* et placer les fichiers générés dans le répertoire *html*, on écrira:

```
<xtask style="page.xsl" todir="html">
```

```
  <fileset dir="xml" includes="*.xml"/>
</xtask>
```

Note

Cette tâche était auparavant distribuée indépendemment sur ma page (en version 0.1). Elle est maintenant partie intégrante de SAT.

Valid

Description

Cette tâche permet de valider des fichiers XML. Elle offre la possibilité de valider la conformité du document à une DTD ou bien de ne vérifier que la syntaxe XML (que le document est *bien formé*). On peut choisir d'arrêter la compilation Ant sur une erreur, le niveau d'erreur ou encore le nombre maximal d'erreurs à afficher par fichier.

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="valid" classname="net.cafebabe.sat.xml.ValidTask"/>
```

Paramètres

Attribut	**Description**	**Requis**
file	Fichier(s) à valider.	Non
dir	Répertoire(s) des fichiers à valider.	Non
dtd	Indique si l'on doit valider la conformité du fichier à une DTD (indiquée dans le `DOCTYPE`) ou si l'on doit se contenter de vérifier qu'il est bien formé (valeur *false*).	Non (*true* par défaut)
failonerror	Indique si le traitement du fichier Ant doit être arrêté après avoir rencontré une erreur (valeur *true*).	Non (*true* par défaut)
errorlevel	Indique au parser le niveau de ce que l'on doit considérer comme des erreurs (qui doivent être affichées et éventuellement arrêter le traitement du fichier Ant). Les valeurs possibles sont *warning*, *error* et *fatal*.	Non (*fatal* par défaut)
maxerrors	C'est le nombre maximal d'erreurs à afficher par fichier validé. Ce nombre doit être supérieur à 0.	Non (*100* par défaut)

Éléments encapsulés

On peut sélectionner un ou plusieurs fichiers XML à valider à l'aide d'un élément `<fileset>`. Voire la documentation de Ant pour plus de détails sur cet élément.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Pour valider les fichiers XML d'un répertoire, recopier dans ce dernier le buildfile suivant:

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

<project name="xml" default="valid" basedir=".">

 <taskdef name="valid" classname="net.cafebabe.sat.xml.ValidTask"/>

 <target name="valid">
  <valid dtd="true"
         failonerror="true"
         errorlevel="fatal"
         maxerrors="100">
   <fileset dir="." includes="*.xml"/>
  </valid>
 </target>

</project>
```

Puis lancer Ant (taper `ant` sur la ligne de commande). On notera que les attributs de la tâche de validation sont ceux par défaut. Adapter ce fichier pour vos besoins.

Si l'on souhaite valider les fichiers Ant de son disque, on lancera à la racine:

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

<project name="xml" default="valid" basedir=".">

 <taskdef name="valid" classname="net.cafebabe.sat.xml.ValidTask"/>

 <target name="valid">
  <valid dtd="false"
         failonerror="false"
         errorlevel="warning"
         maxerrors="1">
   <fileset dir="." includes="**/build.xml"/>
  </valid>
 </target>

</project>
```

L'attribut `dtd` est sur *false* car les buildfile sont réputés ne pas avoir de DTD, on demande à ne pas arrêter le parsing lorsque l'on rencontre des erreurs de manière à scanner tout le disque, on place le niveau d'erreur au minimum de manière à détecter tous les problèmes et enfin, on demande à n'afficher qu'une erreur de manière à ne pas être noyé sous le nombre d'erreurs (qui peut être très important).

Merge

Description

Cet élément permet de fusionner des fichiers XML pour n'en former qu'un (contenant les éléments racine des fichiers fusionnés dans un élément englobant).

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="merge" classname="net.cafebabe.sat.xml.MergeTask"/>
```

Par exemple, cet élément permet de fusionner les deux documents XML suivants:

```
<?xml version="1.0"?>
<racine>
  <element1/>
  <element2/>
  <element3/>
</racine>
```

et:

```
<?xml version="1.0"?>
<racine>
  <element4/>
  <element5/>
</racine>
```

En un seul fichier ci-dessous:

```
<?xml version="1.0"?>
<index>
  <racine>
    <element1/>
    <element2/>
    <element3/>
  </racine>
  <racine>
    <element4/>
    <element5/>
  </racine>
</index>
```

Cet élément est utile pour générer des documents composites issus de la fusion d'autres documents XML. Par exemple, c'est le cas de la page d'accueil de mon site. Elle comporte un texte d'introduction, des nouvelles brèves et des liens qui sont de petits fichiers fusionnés en un seul. Ce fichier résultant est ensuite transformé en HTML (en utilisant XSLT) pour donner la page d'accueil.

Paramètres

Attribut	Description	Requis
file	Fichier(s) à fusionner.	Non
dir	Répertoire(s) de fichiers à fusionner.	Non
tofile	Nom du fichier composite à générer.	Non (*index.xml* par défaut)
encoding	C'est l'encodage du fichier résultant. Cet encodage est iso-8859-1 pour les langues d'Europe de l'ouest, donnant l'élément de déclaration XML suivant `<?xml version="1.0" encoding="iso-8859-1"?>`.	Non (encodage ASCII par défaut)
doctype	C'est la deuxième partie de la déclaration du type de document. Par exemple, un doctype `article PUBLIC "-//CASA//DTD article//FR" "article.dtd"` donnera la déclaration de type de document *<!DOCTYPE article PUBLIC "-//CASA//DTD article//FR" "article.dtd">*. L'élément racine est déduit du premier mot de ce doctype. En l'absence de doctype, l'élément racine est `<index>`	Non (pas de doctype par défaut)

Éléments encapsulés

On peut sélectionner un ou des fichiers XML à fusionner à l'aide d'un élément `<fileset>`. Voire la documentation de Ant pour plus de détails sur cet élément.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Pour fusionner les fichiers XML du répertoire *index* d'encodage *ISO 8859-1* en un fichier *index.xml* de type *index*, on utilisera l'élément suivant:

```
<merge dir="index"
       tofile="index.xml"
       encoding="iso-8859-1"
       doctype="index PUBLIC '-//CASA//DTD index//FR' 'index.dtd'"/>
```

Pour fusionner tous les fichiers XML du répertoire *xml* et de ses sous répertoires en un fichier *index.xml*, on pourra écrire:

```
<merge tofile="index.xml"
       encoding="iso-8859-1"
       doctype="index PUBLIC '-//CASA//DTD index//FR' 'index.dtd'">
  <fileset dir="xml" includes="**/*.xml"/>
</merge>
```

Insert

Description

Cette tâche permet de remplacer une instruction de traitement par le contenu d'un fichier. On peut ainsi inclure un fragment de fichier dans une page HTML par exemple. C'est ainsi que les menus sont inclus dans les pages de mon site.

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="insert" classname="net.cafebabe.sat.xml.InsertTask"/>
```

Paramètres

Attribut	Description	Requis
file	Fichier(s) à traiter.	Non
dir	Répertoire(s) des fichiers à traiter	Non
pattern	C'est le nom de l'instruction de traitement à remplacer. Par exemple si ce pattern est *foo*, alors les instructions de traitement remplacées seront `<?foo ?>`.	Non (*insert* par défaut)
source	Fichier à inclure à la place des instructions de traitement.	Oui

Éléments encapsulés

On peut sélectionner les fichiers à traiter à l'aide d'éléments `<fileset>` encapsulés. Voire la documentation de Ant pour de plus amples informations.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Pour remplacer les instructions de traitement `<?menu ?>` des fichiers du répertoire *html* par le contenu du fichier *menu.html*, on écrira l'élément suivant:

```
<insert dir="html"
        pattern="menu"
        source="menu.html"/>
```

Nest

Description

Cette tâche permet d'encapsuler des fichiers dans un fichier, à l'emplacement d'une instruction de traitement. On peut ainsi inclure des fragments de HTML dans un même corps de document.

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="nest" classname="net.cafebabe.sat.xml.NestTask"/>
```

Paramètres

Attribut	Description	Requis
file	Fichier(s) à traiter.	Non
dir	Répertoire(s) des fichiers à traiter	Non
pattern	C'est le nom de l'instruction de traitement à remplacer. Par exemple si ce pattern est *foo*, alors les instructions de traitement remplacées seront `<?foo ?>`.	Non (*nest* par défaut)
source	Fichier contenant l'instruction de traitement à remplacer par les fichiers.	Oui

Éléments encapsulés

On peut sélectionner les fichiers à traiter à l'aide d'éléments `<fileset>` encapsulés. Voire la documentation de Ant pour de plus amples informations.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Pour encapsuler les fichiers du répertoire *html* dans le fichier *page.html*, à l'emplacement de l'instruction de traitement `<?body ?>`, on écrira l'élément suivant :

```
<nest source="page.html"
      pattern="body"
      dir="html"/>
```

Split

Description

Cette tâche permet de découper des fichiers selon des instructions de traitement. Le nom des fichiers générés est indiqué dans les instructions de traitement délimitant le fichier.

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="split" classname="net.cafebabe.sat.xml.SplitTask"/>
```

Par exemple, le fichier suivant:

```
<?xml version="1.0"?>
<?split file="page1.html"?>
<html>
...
</html>
<?split file="page1.html"?>
<?split file="page2.html"?>
<html>
...
</html>
<?split file="page2.html"?>
```

Sera découpé en deux fichiers (*page1.html* et *page2.html*):

```
<html>
...
</html>
```

Si la plupart des processeurs XSLT (comme XT ou Xalan) proposent des extensions pour générer plusieurs documents, cette tâche permet de s'abstraire de ces particularités et de générer des documents multiples sans polluer son code XSLT de particularités propre à l'un ou l'autre de ces processeurs.

Paramètres

Attribut	Description	Requis
file	Fichier(s) à traiter.	Non
dir	Répertoire(s) des fichiers à traiter	Non
pattern	Instruction de traitement où s'opère le découpage. Par exemple si ce pattern est *split*, alors les instructions de traitement où se fait la découpe seront `<?split file="foo"?>`.	Non (*split* par défaut)

Éléments encapsulés

On peut sélectionner les fichiers à traiter à l'aide d'éléments `<fileset>` encapsulés. Voire la documentation de Ant pour de plus amples précisions.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Pour découper un fichier *foo* selon les instructions de traitement de la forme `<?decoupe file="bar"?>`, on écrira l'élément suivant:

```
<split file="foo"
       pattern="decoupe"/>
```

XML Word Count

Description

Cette tâche compte les mots de documents XML. Très utile lorsqu'on écrit un article calibré. Par défaut, le texte contenu dans tout élément du document est pris en compte alors que le texte contenu dans les attributs est ignoré. Il est possible de changer ce comportement par défaut (voir les paramètres de la tâche).

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="xwc" classname="net.cafebabe.sat.xml.WordCountTask"/>
```

Paramètres

Attribut	**Description**	**Requis**
separators	La liste des caractères de séparation des mots.	Non (a une valeur raisonnable par défaut).
excludeElements	Liste des éléments à exclure du décompte.	Non (par défaut, aucun élément n'est exclu du décompte.
singleElements	Liste des éléments ne comptant que comme un seul mot.	Non (par défaut, aucun élément ne compte comme un mot unique.
includeAttributes	Liste des attributs à inclure au décompte. Ces attributs sont notés sous la forme *element@attribut*, et sont ainsi associés à un élément donné.	Non (par défaut, aucun attribut n'intervient dans le décompte).
documentProperties	Indique un fichier de propriétés des documents (c'est à dire une liste des éléments à exclure, ceux comptant comme un mot unique et la liste des attributs à inclure au décompte). C'est un fichier de propriétés contenant les propriétés `separators`, `excludeElements`, `singleElements` et `includeAttributes`. Chacune de ces propriétés pouvant être vide.	Non (aucun fichier de propriétés n'est chargé par défaut).
property	La propriété Ant à renseigner avec le décompte.	Oui
propertyFiles	La propriété Ant à renseigner avec le nombre de fichiers parsés.	Non (pas de propriété

		renseignée par défaut).
quiet	Si sa valeur est *yes*, la tâche n'affiche pas le décompte des mots.	Non (la valeur par défaut est *no*).

Éléments encapsulés

On peut sélectionner les fichiers à traiter à l'aide d'éléments `<fileset>` encapsulés. Voire la documentation de Ant pour de plus amples précisions.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Supposons que nous voulions compter le nombre de mots des fichiers XML du répertoire courant, sauf le fichier *build.xml* en excluant l'élément `source` du décompte. Nous pouvons écrire la target suivante :

```
<target name="wc">
 <xwc excludeElements="source">
  <fileset dir="." includes="*.xml" excludes="build.xml"/>
 </xwc>
</target>
```

Ce qui produira la sortie suivante :

```
$ ant wc
Buildfile: build.xml

wc:
      [xwc] 8414 words in 2 file(s).

BUILD SUCCESSFUL
Total time: 4 seconds
```

Supposons que nous voulions indiquer les règles du décompte dans un fichier de propriétés, *document.properties* comme suit :

```
excludeElements=comment,source
singleElements=file,keyb,code
includeAttributes=sect@title
```

Ce fichier indique les règles suivantes :

- On doit exclure les éléments `comment` et `source` du décompte.

- Les éléments `file`, `keyb` et `code` ne doivent compter que pour un seul mot dans le décompte.
- Le texte des attributs `title` des éléments `sect` soit être pris en compte.

Pour compter les mots des documents XML du répertoire *xml* en utilisant ces règles et placer le résultat dans la propriété Ant `wc`, on pourra écrire :

```
<target name="wc">
 <xwc property="wc"
      documentProperties="document.properties">
  <fileset dir="xml" includes="*.xml"/>
 </xwc>
</target>
```

Link

Description

Cette tâche permet de vérifier la validité des liens de fichiers HTML. Elle distingue deux types de liens: les liens *locaux* (qui appartiennent à un même site et sont relatifs) et les liens *externes* (qui pointent vers d'autres sites et qui commencent par *http:*, *ftp:* ou autre déclaration de protocole).

La déclaration de la tâche (sous l'élément `<project>`) est la suivante:

```
<taskdef name="link" classname="net.cafebabe.sat.html.LinkTask"/>
```

Paramètres

Attribut	**Description**	**Requis**
file	Fichier(s) à traiter.	Non
dir	Répertoire(s) des fichiers à traiter	Non
external	Indique si l'on doit vérifier les liens externes.	Non (*false* par défaut).
interrupt	Indique si l'on doit interrompre avec une erreur en cas de lien brisé.	Non (*false* par défaut).
log	Indique le nom du fichier dans lequel on souhaite enregistrer le récapitulatif de la vérification des liens.	Non

Éléments encapsulés

On peut sélectionner les fichiers à traiter à l'aide d'éléments `<fileset>` encapsulés. Voire la documentation de Ant pour de plus amples précisions.

Cet élément ne prend pas en compte de contenu textuel.

Exemples

Supposons que l'on génère des pages HTML dans le répertoire *html* et ses sous répertoires. On veut vérifier les liens internes de ce site et interrompre la compilation en cas d'erreur. On pourra écrire:

```
<link external="false"
      interrupt="true">
 <fileset dir="html" includes="**/*.html"/>
</link>
```

En cas de lien brisé, la compilation sera interrompue avec un message indiquant le fichier concerné et le lien fautif.

On veut maintenant vérifier les liens externes (ce qui est bien plus long) et ne pas interrompre la compilation en cas de lien brisé, mais les lister dans un fichier (*links.txt*). On pourra écrire:

```
<link external="true"
      interrupt="false"
      log="links.txt">
 <fileset dir="html" includes="**/*.html"/>
</link>
```

Mode Emacs

Ant est généralement utilisé pour générer des projets Java et est intégré de ce fait dans la plupart des outils de développement Java (comme JDE, un mode Emacs pour le développement Java). Il peut cependant être très utile en d'autres occasions, par exemple pour générer des documents HTML à partir de sources XML. C'est pourquoi j'ai développé ce mode qui permet de lancer Ant pour générer tout type de projet.

Pour installer ce mode Ant, copier le fichier *el/ant.el* dans le répertoire de votre choix où Emacs sera capable de le retrouver, puis ajouter les lignes suivantes à votre fichier de configuration *.emacs* :

```
(load "ant")
(ant-mode)
```

Lorsque vous relancez Emacs, il apparait un menu `Ant` comportant les entrées suivantes :

- **Build**: Permet de lancer Ant. Une invite dans le minibuffer vous donne l'occasion de saisir la cible à lancer. Si vous ne donnez pas de cible, c'est la cible par défault du buildfile qui est exécutée. Il est possible de passer plusieurs cibles en les séparant par des espaces.
- **Rebuild**: Relance Ant avec la cible spécifiée lors de sa dernière exécution.
- **Build File**: Par défaut, le buildfile est recherché récursivement en remontant dans l'arborescence des répertoires. Avec cette option, il est possible d'idiquer un buildfile quelconque.
- **Targets**: Inscrit dans le minibuffer la liste des cibles du buildfile.
- **Help**: Affiche une aide sur le buildfile (la liste des cibles et leur description).

- **Start Server**: Lance Ant en mode server (voir la tâche `server` ci-dessus). Cette fonctionalité est encore expérimentale et passablement buggée.
- **Stop Server**: Arrête Ant lancé en mode server.

Le résultat du build est affiché dans un buffer de compilation, ce qui permet de cliquer sur les erreurs pour ouvrir le fichier fautif dans Emacs.

Il peut être très commode d'affecter ces appels à Ant à des touches du clavier. Par exemple, mon fichier *.emacs* comporte les affectations suivantes :

```
(global-set-key [f2] 'ant-build)
(global-set-key [f3] 'ant-rebuild)
```

Licence

Ce logiciel est distribué sous licence Apache Software License. Vous trouverez une copie de cette licence dans le répertoire d'installation de SAT, dans le fichier *LICENSE*.

Historique

Version 1.0 (2003-07-15)

Les modifications sont les suivantes :

- La tache `valid` a été munie d'un nouvel attribut `reference` (fichier servant de référence pour les dates : un fichier n'est validé que s'il est plus récent que la régérence, cette référence est mise à jour après chaque validation). On évite ainsi de valider inutilement des fichiers. Attention, si la DTD a changée, le système ne peut le savoir et un fichier XML peut ne pas etre validé dans ce cas. Pour etre certain que tous les fichiers seront validés, ne pas renseigner cet attribut.
- La tache `insert` a été modifiée de manière à ce qu'elle n'insère le fichier source que dans les fichiers plus récents que la dernière insertion (nouvel attribut référence dans la tache `insert`).

Version 0.9 (2003-07-14)

Les modifications sont les suivantes :

- La tache `server` a été supprimée du projet et ne sera plus supportée. Utiliser la Console Ant à la place (plus simple à mettre en oeuvre).

Version 0.8 (2003-07-13)

Les modifications sont les suivantes :

- Toutes les tâches ont changé de package (pour passer dans `net.cafebabe.sat`). C'est la dernière fois, promis juré :o)

- Développement de la console (qui est sensiblement plus efficace que le mode serveur).
- Scripts *antc* pour lancer la console et *ants* pour lancer le mode serveur.

Version 0.7 (2002-11-11)

La documentation a été passée à ma nouvelle DTD et mise à jour.

Nouvelles tâches :

- Server: Accélère les builds.
- Nest: Nouvelle tâche de manipulation XML.
- XML Word Count: Pour compter les mots d'un document XML.

Corrections et améliorations multiples :

- Débuggage de la tâche XTask (pour faire tourner en mode serveur).
- Amélioration de la gestion de l'encodage de la tâche `merge`.
- Mise à jour pour Ant version 1.5.1.

Version 0.6 (2001-10-27)

Optimisation de la tâche `Merge` (par utilisation de `StringBuffers`). On peut espérer un gain de performances d'un facteur 30 !

Version 0.5 (2001-10-17)

Nouvelle tâche `Valid` pour validation des fichiers XML.

Version 0.4 (2001-09-22)

Nouvelle tâche `Link` permettant de tester les liens de fichiers HTML.

Version 0.3 (2001-09-18)

Test des attributs des tâches. Les attributs de fichier (`file` et `dir`) sont testés.

Correction de bugs (les chemins des fichiers sont maintenant relatifs au répertoire du fichier de projet).

Dans une tâche `BSH`, on peut maintenant lire et écrire des propriétés Ant avec les commandes `setAntProperty()` et `getAntProperty()`.

Les tâches ne sont plus exécutables en tant que programme indépendant (il est beaucoup plus pratique des les exécuter dans Ant et c'est très pénible à maintenir).

Le code a été adapté aux dernières versions de Beanshell (version 1.1a18 supportée, la 1.2b1 ne fonctionne pas à cause d'un bug).

Implémentation de tests unitaires (dans le répertoire *prj/test*).

Version 0.2 (2001-08-20)

Adoption de l'API JAXP pour la tâche XTask. Permet d'instancier le parser XML de manière générique (la tâche est maintenant indépendante du parser SAX utilisé pourvu qu'il implémente l'API JAXP).

Version 0.1 (2001-06-06)

Les tâches ont été remaniées (après usage intensif pour construire mon site) de manière à être plus efficaces/ergonomiques. Les changements sont les suivants:

- `bsh`: Les éléments encapsulés `<arg>` ont disparus (on initialise des variables dans un script encapsulé). L'attribut `args` est apparu pour passer des arguments comme s'ils avaient été passés sur la ligne de commande. L'attribut `source` a été remplacé par `file`. Apparition de l'attribut `dir`. Implémentation de l'élément `<fileset>`. Redirection de la sortie des scripts de manière standard (après un marqueur [bsh]).
- `xtask`: Correction de bug. Refonte complète des attributs (peu utilisables dans la version précédente). Implémentation de l'élément `<fileset>`.
- `merge`, `insert` et `split`: harmonisation avec les attributs `file` et `dir`. Attribut verbose supprimé (optimisation des messages). Implémentation de l'élément `<fileset>`.

Il ne devrait plus intervenir de changements concernant la syntaxe des tâches. Les versions futures ne feront que stabiliser le code (il reste à tester intensivement es tâches et à coder des tests sur les attributs des tâches pour afficher des messages d'erreur clairs).

Version 0.0 (2001-05-28)

Première version de SAT. Comporte les tâches `Bsh`, `XTask`, `Merge`, `Insert` et `Split`.

Enjoy !