BeanShellEditor for JUMP

version 0.1.0

Résumé

BeanShellEditor est un éditeur de script BeanShell gratuit. JUMP est un SIG OpenSource. Cet article explique comment utiliser BeanShellEditor comme PlugIn de JUMP.

Table of Contents

Comment installer le PlugIn BeanShellEditor

De quoi avez-vous besoin ?

Pour utiliser BeanShellEditor comme PlugIn de JUMP vous aurez besoin :

- D'une machine virtuelle java (JVM) 1.4+ installée sur votre système (téléchargeable sur le site de sun à java [http://java.com/en/]).
- JUMP : le projet principal est hébergé par http://www.jump-project.org/ et plusieurs branches intéressantes peuvent être trouvées de par le monde :
  - JPP [http://jump-pilot.sourceforge.net/]
  - SIGLE [http://www.projet-sigle.org/]
  - AGILES [http://www.agiles.org/]
- BeanShellEditor : vous devriez trouver le programme BeanShellEditor pour JUMP avec cette documentation. Sinon, rendez vous sur le site SIGLE [http://www.projet-sigle.org/] .
- Si vous ne les avez pas récupéré avec la distribution, vous pouvez trouver les packages BeanShell et

Buoy aux adresses suivantes

- BeanShell [http://www.beanshell.org]
- Buoy [http://buoy.sourceforge.net/]

Votre répertoire d'installation

Votre répertoire d'installation devrait ressembler à quelquechose comme :

```
+ JUMP
  jumpworkbench.bat (for windows)
  workbench-properties.xml
  + lib
    jump-1_0.jar
    jts-1.4.1-RC1.jar
    bsh-2.0b2.jar
    Jama-1.0.1.jar
    jts-1.4.1-RC1.jar
    log4j-1.2.8.jar
    jdom.jar
    xercesImpl.jar
    xml-apis.jar
    + ext
      bsheditor.jar
      buoy.jar
      ... (other plugins)
```

Warning

Pour tirer pleinement partie de BeanShell, vous devez installer la dernière version de la librairie bsh-2.0b2 (le fichier inclu dans la version 1.1.2 de JUMP est bsh-2.0b1) .

Par défaut, Jump charge tous les plugins situés dans le répertoire `ext`. Le PlugIn BeanShellEditor crée l'item Bean Shell Editor dans le menu Scripting :

Premiers pas avec BeanShellEditor

Dans cette section, nous allons découvrir l'API du JUMP à travers de petits scripts BeanShell.

Pour commencer, ouvrez BeanShellEditor à l'aide du menu Scripting / BeanShell Editor.

Tous les objects de votre application sont accessibles via l'objet WorkbenchContext passé en paramètre du PlugIn sous le nom de "wc" (ce nom peu romantique en français assure la compatibilité avec les plugins développés via le PlugIn BeanShell de la distribution originale).

Tip

Vous pouvez faire du copier/coller à partir des exemples de ce document, lancer les scripts, et sauver dans un répertoire dédié ceux que vous voulez garder ou modifier plus tard.

.

Imprimer des informations diverses sur les couches

Imprimer la liste complète des layers

Dans l'éditeur, écrivez :

```
// La liste des couches est obtenue du layerManager
layers = wc.getLayerManager().getLayers();
// Ecrire le nom de chaque couche
for(i=0 ; i<layers.size() ; i++) print(layers.get(i));
```

Puis, pressez le bouton RUN situé au-dessus de l'éditeur. Vous devez obtenir la liste des couches dans la fenêtre de sortie de BeanShellEditor.
Avec BeanShell 2.0, vous pouvez utiliser la syntaxe simplifiée suivante pour les accesseurs et pour les boucles :

```
// La liste des couches est obtenue du layerManager
layers = wc.layerManager.layers;
// Ecrire le nom de chaque couche
for(layer:layers) print(layer);
```

Vous pouvez même écrire un tableau ou une collection à l'aide d'une commande d'une seule ligne (et utiliser pour celà la ligne de commande) :

```
print(wc.layerManager.layers);
```

Un tableau ou une collection sont imprimés par beanshell comme une liste entre crochets, de valeurs séparées par des virgules :

[COUNTRY, CITY, RIVER, ROAD]

Imprimer le schéma de données des couches sélectionnées

Pour imprimer le schéma associé à chaque couche, vous pouvez écrire :

```
//************************************************
// IMPRESSION DES SCHEMAS DES COUCHES SELECTIONNEES
//************************************************
// Les couches sélectionnées sont obtenues du LayerNamePanel
layers = wc.getLayerNamePanel().getSelectedLayers();
// Pour chaque couche, imprimer le nom de la couche et son schema
// Noter qu'ici, layers représente un Array (voir l'API JUMP)
for(i=0 ; i<layers.length ; i++) {
    print(layers[i]);
    schema = layers[i].getFeatureCollectionWrapper()
                      .getFeatureSchema();
    for (j=0 ; j<schema.getAttributeCount() ; j++) {
        print("\t" + schema.getAttributeName(j) +
              "\t" + schema.getAttributeType(j));
    }
}
```

Tip

En java 1.4+, on utilise une syntaxe différente pour parcourir un tableau (Array) ou une collection :

```
for(i=0 ; i<list.size() ; i++) print(list.get(i));

for(i=0 ; i<array.length ; i++) print(array[i]);

Avec java 5.0 et beanshell 2.0, vous pouvez utiliser la même syntaxe
pour les deux formes de boucle :

for (i : list_or_array) print(i);
```

Imprimez le nombre d'objets de chaque couche dans la fenêtre de sortie de JUMP (en HTML)

Pour créer un nouveau document HTML dans la fenêtre de sortie de JUMP, vous devez juste écrire :

```
wc.getWorkbench().getFrame().getOutputFrame();

htmlFrame.createNewDocument();
```

Vous pouvez maintenant écrire dans ce `htmlFrame` en utilisant les balises html usuelles :

```
//************************************************
// IMPRIME DANS LA FENETRE DE SORTIE DE JUMP UN
// TABLEAU HTML CONTENANT LE NOM DES LAYERS ET LE
// NOMBRE D'OBJETS QU'ILS CONTIENNENT
//************************************************
// Obtenir la fenetre de sortie
htmlFrame = wc.getWorkbench().getFrame().getOutputFrame();
htmlFrame.createNewDocument();

// Initialiser la table
htmlFrame.append("<h2>NUMBER OF FEATURES IN LAYERS</h2>");
htmlFrame.append("<table frame=\"box\" border=\"3\">");
htmlFrame.append("<th>Layer name</th><th>Number of features</th>");

// Imprimer les résultats
for(layer : wc.getLayerManager().getLayers()) {
    htmlFrame.append("<tr><td>" + layer + "</td><td>" +
                     layer.getFeatureCollectionWrapper().size() +
                     "</td></tr>");
}
htmlFrame.append("</table>");

// Passer la fenetre de sortie sur le devant
htmlFrame.surface();
```

Travailler avec les couches (layer)

Créer une nouvelle couche vide à partir d'une couche existante

Ici, nous allons créer une nouvelle couche vide à partir du schéma de données d'une autre couche présente :

```
//************************************************
// CREER UNE COUCHE VIDE
```

```
// A PARTIR DU SCHEMA DE LA COUCHE SELECTIONNEE
// (COMMENCER PAR SELECTIONNER UNE COUCHE)
//************************************************

import com.vividsolutions.jump.feature.*;

layers = wc.getLayerNamePanel().getSelectedLayers();

// Une couche doit être selectionnée
if(layers.length==0) {
    wc.getWorkbench().getFrame().warnUser("A layer must be selected !");
}
else if (layers.length>1) {
    wc.getWorkbench().getFrame().warnUser("Only 1 layer must be selected !");
}
else {
    layer = layers[0];
    featureSchema = layer.getFeatureCollectionWrapper()
                         .getFeatureSchema();
    wc.getLayerManager().addLayer("NewCategory",
                                  "EmptyLayerFrom_"+layer.getName(),
                                  new FeatureDataset(featureSchema));
}
```

Si vous créez des couches vides à partir de toutes les couches de votre tâche active, enlevez le second test du code ci-dessus et incluez-le dans la boucle suivante :

```
for (layer : layers) {
    featureSchema = layer.getFeatureCollectionWrapper().getFeatureSchema();
    wc.getLayerManager().addLayer("NewCategory",
                                  "EmptyLayerFrom_" + layer.getName(),
                                  new FeatureDataset(featureSchema));
}
```

Créer une couche avec un schéma de données spécifique

```
//*****************************************************
// CREER UNE COUCHE AVEC UN SCHEMA DE DONNEES SUR MESURE
//*****************************************************
import com.vividsolutions.jump.feature.*;

FeatureSchema fs = new FeatureSchema();
fs.addAttribute("GEOMETRY", AttributeType.GEOMETRY);
fs.addAttribute("Name", AttributeType.STRING);
fs.addAttribute("Population", AttributeType.INTEGER);
fs.addAttribute("Capital", AttributeType.STRING);

wc.getLayerManager()
  .addLayer("Working", "MyNewLayerName", new FeatureDataset(fs));
```

Créer une nouvelle couche à partir d'une sélection

Ici, nous allons sélectionner par programme une partie des objets de la couche sélectionnée, et les copier vers une nouvelle couche. Ce script nécessite de sélectionner une couche particulière contenant un attribut numérique appelé "Population".

Note

Dans les paragraphes §2.3 et § 3, nous apprendrons comment inclure ce type de code dans des méthodes

ou des classes plus génériques.

```
//*************************************************
// CREER UNE NOUVELLE COUCHE A PARTIR D'UNE SELECTION
// Pour exécuter ce script, il doit y avoir un attribut
// numérique "Population" dans la couche sélectionnée
//*************************************************
import com.vividsolutions.jump.feature.*;

// Nom de votre sélection (qui sera le nom de la couche)
selection = "Country50M";

// Obtient la couche sélectionnée
layers = wc.getLayerNamePanel().getSelectedLayers();

if(layers.length!=1) {
    wc.getWorkbench().getFrame()
      .warnUser("The number of selected layers must be exactly one !");
}
else {
    // Get the feature collection displayed by the selected layer
    featureCollection = layers[0].getFeatureCollectionWrapper();
    // Create a new dataset (feature collection) with the same schema
    featureDataset =
        new FeatureDataset(featureCollection.getFeatureSchema());
    // Test all the features of the collection with an iterator
    for (it = featureCollection.iterator() ; it.hasNext() ; ) {
        feature = it.next();
        if(feature.getAttribute("Population")>=50000000) {
            featureDataset.add(feature);
        }
    }
}

// Créer la nouvelle couche à partir de la sélection
// remarque : les objets de la nouvelle couche sont
// des copies complètes des originaux
wc.getLayerManager().addLayer("Working", selection, featureDataset);
```

Créer une nouvelle couche par type de géométrie

Certains modèles de données nécessitent des types géométriques homogènes. Voici un script qui va répartir les objets dans différentes couches en fonction de leur type géométrique.

```
//*********************************************
// CREER UNE COUCHE PAR TYPE GEOMETRIQUE
// Une couche doit etre sélectionnée
// Même les couches devant rester vide sont créées
//*********************************************
import com.vividsolutions.jump.feature.*;

// Couche sélectionnée
layers = wc.getLayerNamePanel().getSelectedLayers();

if(layers.length!=1) {
    wc.getWorkbench()
      .getFrame()
      .warnUser("The number of selected layers must be exactly one !");
}
else {

    // FeatureCollection pointée par la couche sélectionnée
```

```
    layer = layers[0];
    fc = layer.getFeatureCollectionWrapper();
    fs = fc.getFeatureSchema();

    // Crée les layers dérivés du layer d'origine;
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_Point", new FeatureDataset(fs));
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_LineString", new FeatureDataset(fs));
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_Polygon", new FeatureDataset(fs));
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_MultiPoint", new FeatureDataset(fs));
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_MultiLineString", new FeatureDataset(fs));
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_MultiPolygon", new FeatureDataset(fs));
    wc.getLayerManager().addLayer("Working",
        layer.getName()+"_GeometryCollection", new FeatureDataset(fs));

    // Passe en revue tous les objets à l'aide d'un Iterator
    for (it = fc.iterator() ; it.hasNext() ; ) {
        feature = it.next();
        wc.getLayerManager()
          .getLayer(layer.getName()+"_"+feature.getGeometry().getGeometryType())
          .getFeatureCollectionWrapper()
          .add(feature);
    }
}
```

Créer de nouvelles méthodes, de nouvelles classes, de nouveaux threads

Créer une classe mesurant la surface des objets sélectionnés

Ici, nous allons simplement écrire la fonction dans l'éditeur, puis utiliser celle-ci en ligne de commandes :

```
//*******************************
// SURFACE DES OBJETS SELECTIONNES
//*******************************
area() {
    items = wc.getLayerViewPanel.getSelectionManager().getSelectedItems();
    double area = 0.0;
    for(item : items) area += item.getArea();
    print(java.text.NumberFormat.getInstance().format(area));
    return area;
}
```

Executez le script, puis essayez le plusieurs fois sur différentes sélections, en utilisant la ligne de commande :

```
area();
```

Vous pouvez également utiliser le panneau variables & methods et cliquer sur le nom de la méthode (`#area`) pour l'écrire dans la zone de texte de la ligne de commande sans vous tromper. N'oubliez pas de rajouter un ";" à la fin.

Créer une interface représentant une condition pour effectuer des requêtes

Dans le script suivant, la condition est une instance de classe implémentant l'interface Condition et passée en paramètre de la requête.

Warning

Pour exécuter ce code, il est important de disposer de la version `bsh-2.0b2.jar` de BeanShell dans le classpath (plutôt que la version bsh-2.0b1.jar distribuée avec jump 1.1.2). Vous pouvez également être amenés à modifier le classpath dans le lanceur de l'application (fichier bat pour windows par exemple).

```
//************************************************
// IMPLEMENTER L'INTERFACE CONDITION POUR EXECUTER
// UNE REQUETE SUR UNE COUCHE
// Ce script n'exécute pas explicitement une requête
// il définit le cadre pour l'exécution d'une requête
// (voir aussi le script suivant)
//************************************************

import com.vividsolutions.jump.feature.*;
import com.vividsolutions.jump.workbench.model.*;
import com.vividsolutions.jump.workbench.ui.InfoFrame;

// INTERFACE POUR UNE CONDITION
public interface Condition {public boolean test(Feature f);}


// METHODE DEFINISSANT UNE REQUETE BASEE SUR UNE CONDITION
// ET RENVOYANT UN OBJET DISPOSANT DE 3 METHODES UTILES
// - getResult() renvoie le jeu de données répondant à la requête
// - createNewLayer() crée une couche contenant le résultat de la requête
// - getAttributePanel() affiche la table attributaire de la requête
query(Layer layer, Condition condition) {
    this.layer = layer;
    fc = layer.featureCollectionWrapper;
    dataset = new FeatureDataset(fc.featureSchema);
    for (it=fc.iterator() ; it.hasNext() ; ) {
        f = it.next();
        if (condition.test(f)) {dataset.add(f);}
    }

    // RETURN THE QUERY AS A FEATURE DATASET
    getResult() {return dataset;}

    // CREATE A NEW LAYER FROM THE QUERY RESULT
    createNewLayer(String category, String layer) {
        if (dataset.size()==0) return;
        wc.layerManager.addLayer(category, layer, dataset);
    }

    // CREATE AND DISPLAY THE ATTRIBUTE TABLE OF THE SELECTION
    getAttributeTable() {
        info = new InfoFrame(wc, wc, wc.workbench.frame.activeInternalFrame);
        info.model.add(layer, dataset.features);
        wc.workbench.frame.addInternalFrame(info);
    }
    return this;
}
```

Et maintenant, vous pouvez utiliser ces objets comme ceci (à la suite du premier script ou en ligne de commande, après l'exécution du script précédent) :

```
// CET EXEMPLE AFFICHE LA TABLE ATTRIBUTAIRE CONTENANT LES
// OBJETS DE MyLayer DE PLUS DE 100 M DE LONG
query(wc.layerManager.getLayer("MyLayer"),
    new Condition(){
        public boolean test(Feature f){return f.geometry.length<100;}
    }
).getAttributeTable();
```

Vous pouvez aussi inclure la définition de l'interface Condition et de la méthode query dans un nouveau script grâce à la commande beanshell `interpreter.source` command (voir § 3.3), voire même l'inclure dans votre script de démarrage afin d'en disposer en permanence (voir § 3.2).

Inclure un script dans un nouveau Thread pour les processus un peu longs.

Voir la documentation de BeanShellEditor pour un exemple.

Utilisation avancée

Ici, nous étudierons la manière de personnaliser l'environnement afin de gagner en efficacité.

Un répertoire dédié à vos scripts

Organisez vos scripts dans un répertoire, par exemple bsh. Ensuite, sélectionner ce répertoire en utilisant le menu option --> script file folder. Vous obtenez ainsi une "vue" de vos scripts sous forme d'arborescence dans le panneau situé à gauche de l'éditeur. L'ouverture de l'un de ces script se fait en double-cliquant sur son nom.

Vous pouvez ouvrir un script (double-click), le modifier, l'exécuter. Si vous voulez sauvegarder vos changements, cliquer simplement sur le bouton sauver ou sauver comme dans la barre d'outils.

Un script de démarrage pour personnaliser votre environnement

Avez-vous remarqué que certains imports, certaines variables ou certaines méthodes étaient appelés presque systématiquement dans vos scripts ? Pour gagner du temps, écrivons un script qui sera toujours appelé et exécuter avant l'exécution d'un nouveau script.

Votre script de démarrage pourrait par exemple ressembler à ceci :

```
//***************************************************************
// SCRIPT DE DEMARRAGE BEANSHELL
// Ce script est interprété chaque fois qu'un nouvel objet interpreter
// est créé par BeanShellEditor (chaque fois qu'un script est exécuté
// à partir de l'éditeur de scripts).
// C'est l'endroit idéal pour :
//    - effectuer des imports
//    - définir des constantes de l'application
//    - définir les fonctions de base
//    - ...
//***************************************************************
```

```
print("Starting script...");
print("The following methods and variables are always available :");

//****************************************************************
// FONCTIONS DE BASE POUR L'IMPESSION DE LA DATE OU DE L'HEURE
//****************************************************************

print("\tdate()");
String date() {new java.text.SimpleDateFormat("dd-MM-yyyy")
               .format(new Date(System.currentTimeMillis()));}

print("\ttime()");
String time() {new java.text.SimpleDateFormat("HH:mm:ss")
               .format(new Date(System.currentTimeMillis()));}

//****************************************************************
// IMPORT DES PACKAGES DE JUMP ET DE JTS
//****************************************************************

import com.vividsolutions.jump.feature.*;
import com.vividsolutions.jump.task.*;
import com.vividsolutions.jump.tools.*;   // OverlayEngine
import com.vividsolutions.jump.ui.*;
import com.vividsolutions.jump.util.*;
import com.vividsolutions.jump.util.java2xml.*;
import com.vividsolutions.jump.workbench.*;
import com.vividsolutions.jump.workbench.model.*;
import com.vividsolutions.jump.workbench.plugin.*;
import com.vividsolutions.jump.workbench.ui.*;
import com.vividsolutions.jump.workbench.ui.plugin.*;
import com.vividsolutions.jump.workbench.ui.wizard.*;

import com.vividsolutions.jts.algorithm.*;
import com.vividsolutions.jts.geom.*;
import com.vividsolutions.jts.geom.util.*;
import com.vividsolutions.jts.geomgraph.*;
import com.vividsolutions.jts.index.strtree.*;
import com.vividsolutions.jts.noding.*;
import com.vividsolutions.jts.operation.buffer.*;
import com.vividsolutions.jts.operation.distance.*;
import com.vividsolutions.jts.operation.linemerge.*;
import com.vividsolutions.jts.operation.overlay.*;
import com.vividsolutions.jts.operation.polygonize.*;
import com.vividsolutions.jts.operation.relate.*;
import com.vividsolutions.jts.operation.valid.*;
import com.vividsolutions.jts.planargraph.*;
import com.vividsolutions.jts.precision.*;
import com.vividsolutions.jts.util.*;

//****************************************************************************
// REFERENCE DE QUELQUES OBJETS ET METHODES UTILES POUR ACCEDER AUX DONNEES
// Note : wc est une variable définie dans le PlugIn BeanShellEditor pour
// JUMP, et référencant l'objet WorkbenchContext.
//****************************************************************************

// Define layerManager
print("\tlayerManager");
LayerManager layerManager = wc.getLayerManager();

// Define layerNamePanel;
print("\tlayerNamePanel");
layerNamePanel = wc.getLayerNamePanel();

// Define method getSelectedLayers();
```

```
print("\tgetSelectedLayers()");
Layer[] getSelectedLayers(){
    layers = wc.layerNamePanel.getSelectedLayers();
    if (layers.length>0) return layers[0];
    else print("One layer must be selected");
}

// Define method getFeatureCollection(String layer)
print("\tgetFeatureCollection(String layer)");
FeatureCollection getFeatureCollection(String layer) {
    return wc.getLayerManager().getLayer(layer)
             .getFeatureCollectionWrapper();
}

// Define method getSelection(String layer) returning a Collection
print("\tgetSelection() --> Collection");
java.util.Collection getSelection(){
    return wc.getLayerViewPanel().getSelectionManager()
             .getFeaturesWithSelectedItems();
}


// Definit une méthode envoyant un avertissement en bas
// à gauche de l'interface JUMP
warning(String s) {wc.getWorkbench().getFrame().warnUser(s);}

// ... add your preferred methods
```

Faire travailler les scripts ensemble

Un script BeanShell peut en appeler un autre (les chemins relatifs des fichiers font alors référence au répertoire d'appel de la JVM) :

```
// Le chemin est relatif au répertoire d'appel
this.interpreter.source("../../bsh/jumptoolbox.bsh");
```

Si vous voulez que votre programme continue sans s'interrompre en cas d'échec de type "fichier non trouvé", vous devez inclure l'appel dans un bloc try/catch :

```
try {this.interpreter.source("../../bsh/jumptoolbox.bsh");}
catch(Exception e) {print(e.getMessage());}
```

Exemple de mise en oeuvre

Dans cet exemple, nous allons séparer le script contenant les méthodes communes pouvant servir dans d'autres contextes, du script faisant appel à ces méthodes pour exécuter un travail bien particulier dans des conditions particulières.

```
//*********************************************
// BOITE A OUTILS "ATTRIBUTS" CONTENANT :
// - une methode trim() enlevant les espaces inutiles
//   sur les attributs texte
// - une methode ajoutant aux objets (et au schema)
//   un attribut issu de la concaténation de deux
//   autres attributs
//*********************************************
import com.vividsolutions.jump.feature.*;
```

```
import com.vividsolutions.jump.workbench.model.*;

// Enlever les espaces en début et en fin de chaîne de caractères
trim(Feature f) {
    fs = f.getSchema();
    for (i=0 ; i<fs.getAttributeCount() ; i++) {
        if (fs.getAttributeType().toJavaClass().equals(String.class)) {
            f.setAttribute(f.getAttribute().trim());
        }
    }
}

// Créer un nouvel attribut String défini par
// attribute 1 + sep + attribute 2
concat(Layer layer, String att1, String att2, String att3, String sep) {
    fc = layer.getFeatureCollectionWrapper();
    fs = fc.getFeatureSchema();
    if (fs.hasAttribute(att1) && fs.hasAttribute(att2)) {
        fs.addAttribute(att3, AttributeType.STRING);
        for (f:fc.getFeatures()) {
            newAttributes = new Object[fs.getAttributeCount()];
            System.arraycopy(f.getAttributes(),
                             0,
                             newAttributes,
                             0,
                             newAttributes.length-1);
            f.setAttributes(newAttributes);
            f.setAttribute(fs.getAttributeIndex(att3),
              f.getString(att1)+sep+f.getString(att2));
        }
    }
    else {
        print("Attribute " + att1 + " or " + att2 + " does not exist");
    }
}
```

Maintenant, sauvegardons le script quelquepart (ex. `bsh/scriptutils.bsh`) et appelons-le à partir d'un autre script spécialement conçu pour traiter une couche ADRESSE (pour tester ce script, vous devrez créer cette couche avec un schéma de données contenant au moins un attribut "NUMBER", et un attribut "STREET", ainsi que quelques données) :

```
// enlève les espaces initiaux et finaux
layer = wc.layerManager.getLayer("ADDRESS");
for (f:layer.getFeatureCollectionWrapper().getFeatures()) trim(f);

// puis ajoute l'attribut ADDRESS
concat(layer, "NUMBER", "STREET", "ADDRESS", " ");
```

Si vous voulez rendre vos scripts plus robuste, n'hésitez pas à envoyer des exception, ou à utiliser la méthode `wc.workbench.frame.warnUser()` pour avertir l'utilisateur que la couche n'existe pas, que les attributs sont incorrects...

Utiliser une bibliothèque tierce

Dans cet exemple, nous allons utiliser la bibliothèque itext (un projet open-source permettant de générer des documents pdf) pour constituer un atlas dont chaque page sera une vue de la tâche active centrée sur un objet différent de la couche de votre choix. Un attribut, également entré en paramètre, servira à titrer les cartes. Typiquement, vous pouvez l'utiliser pour réaliser un atlas des pays (resp. communes à partir de la couche pays (resp. commune).

La première instruction consiste à ajouter la bibliothèque itext au classpath. Si vous l'ajouter dans le répertoire ext des plugin, l'instruction sera :

```
addClassPath("lib/ext/itext-1.02b.jar");


//*************************************************************
// CREE UN SIMPLE ATLAS A PARTIR D'UNE TACHE DE JUMP?
// EN UTILISANT LA BIBLIOTHEQUE ITEXT
//*************************************************************

// Ajouter itext au classpath
// Le fichier jar est supposé être placé
// dans le répertoire d'éxécution du programme
addClassPath("itext-1.02b.jar");

import com.vividsolutions.jump.workbench.ui.LayerPrinter;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import com.lowagie.text.*;
import com.lowagie.text.pdf.*;
import com.lowagie.text.pdf.fonts.*;



// @param title title of the pdf document
// @param layer name of a layer. Each object from this layer
// (ex. country) will be printed out on a new pdf page
// @param attribute layer's attribute used for map titles
// @param filename output file name
atlasPDF(String title, String layer, String attribute, String filename) {
    // Create the document
    Rectangle pageSize = PageSize.A4;
    pageSize.setBackgroundColor(new java.awt.Color(0xFF, 0xFF, 0xDE));
    Document document = new Document(pageSize);
    PdfWriter writer =
        PdfWriter.getInstance(document, new FileOutputStream(filename));

    // Create metadata (must be done before document.open())
    document.addTitle("title");
    document.addSubject("Atlas");
    // ...add other metadata here

    // document opening
    document.open();
    document.resetHeader();

    // Font definition
    BaseFont arial =
        BaseFont.createFont("Helvetica", BaseFont.CP1252, BaseFont.NOT_EMBEDDED);
    Font titlefont = new Font(arial, 40, Font.NORMAL);
    Font imagetitlefont = new Font(arial, 20, Font.NORMAL);

    // Title page
    Paragraph parag = new Paragraph(title, titlefont);
    parag.setAlignment(Paragraph.ALIGN_CENTER);
    document.add(parag);

    fc = wc.layerManager.getLayer(layer).featureCollectionWrapper;
    //images(layer, attribute);
    for (it = fc.iterator() ; it.hasNext() ; ) {
        f = it.next();
        document.newPage();
        Paragraph parag = new Paragraph(f.getAttribute(attribute), imagetitlefont)
```

```
            parag.setAlignment(Paragraph.ALIGN_CENTER);
            document.add(parag);

            printer = new LayerPrinter();
            env = f.getGeometry().getEnvelopeInternal();
            image = printer.print(wc.getLayerManager().getLayers(), env, 400);
            itextimage = com.lowagie.text.Image.getInstance(image, null);
            itextimage.setAlignment(Image.MIDDLE);
            itextimage.scalePercent(100);
            document.add(itextimage);
        }
        document.close();
        writer.close();
}
```

Pour appliquer cette méthode à votre carte, appelez-la de la manière suivante :

```
atlasPDF("WORLD ATLAS", "Country", "Name", "MyAtlas.pdf")
```

Ecrire de nouveaux PlugIn

Voici un court exemple illustrant la manière dont on peut créer un nouveau PlugIn JUMP, et appellant un script BeanShell. Il s'agit là d'une méthode permettant de développer/corriger/modifier ses PlugIn de manière extrêmement facile et efficace.

Tip

Ajoutez la méthode addPlugIn() à votre script de démarrage et développez de nouveaux PlugIns en quelques minutes.

Warning

BeanshellEditor possède aussi une méthode addPlugin qui ajoute un PlugIn à l'éditeur lui-même (et nom à l'application de JUMP).

```
//************************************************
// CETTE METHODE CREE UN NOUVEAU PLUGIN APPELLANT
// LE SCRIPT path.bsh
//************************************************

import com.vividsolutions.jump.workbench.plugin.*;
import com.vividsolutions.jump.task.*;

addJumpPlugIn(String path) {
    final String plugInPath = path;
    // CREE UNE CLASSE IMPLEMENTANT ThreadedPlugIn
    class BshPlugIn implements ThreadedPlugIn {
        // INITIALISE UN MENU BASE SUR LE CHEMIN DU FICHIER BSH
        public void initialize(PlugInContext context) throws Exception {
            String[] sss = plugInPath.split("/");
            String[] ss = new String[sss.length-1];
            System.arraycopy(sss, 0, ss, 0, sss.length-1);
            context.getFeatureInstaller().addMainMenuItem(this,
                ss, sss[sss.length-1], false, null, null
            );
        }
        public boolean execute(PlugInContext context) throws Exception {
            return true;
        }
        // LANCE LE SCRIPT plugInPath+".bsh"
        public void run(TaskMonitor monitor, PlugInContext context) {
```

```
            this.interpreter.set("monitor", monitor);
            try {this.interpreter.source(plugInPath+".bsh");}
            catch(Exception e) {print(e);}
        }
        public String getName() {return plugInPath;}
    }

    //CREE UNE CLASSE INITIALISANT LE PLUGIN
    class MyExtension extends Extension {
        public void configure(PlugInContext context) throws Exception {
            new BshPlugIn().initialize(context);
        }
    }

    // CREE UNE INSTANCE D'EXTENSION (voir API JUMP)
    new MyExtension().configure(wc.createPlugInContext());

}
```

Maintenant, sauvegarder le script et créer le PlugIn JUMP qui va faire le travail (dans le cas présenté ci-dessous, on veut créer un script de fusion de linéaires qui sera appelé par l'élement de menu MyScripts -> lineMerging).

Ce script permet de fusionner les éléments d'une couche d'objets linéaires sur la base d'un attribut (les éléments jointifs ayant même attribut sont fusionnés). Il propose au départ la liste d'attribut disponibles pour la couche sélectionnée :

```
//*************************************
// LINE MERGING PLUGIN
//*************************************

import com.vividsolutions.jump.feature.*;
import com.vividsolutions.jump.workbench.ui.*;
import com.vividsolutions.jts.operation.linemerge.*;

layers = wc.layerNamePanel.selectedLayers;

lineMerging() {
    // EXCEPTION
    if(layers.length!=1) {
        return wc.workbench.frame.warnUser("One layer must be selected !");
    }
    // INITIALISATION
    fc = layers[0].featureCollectionWrapper;
    fs = fc.featureSchema;
    ds = new FeatureDataset(fs);

    // CHOIX DE L'ATTRIBUT DE FUSION
    mid = new MultiInputDialog(wc.workbench.frame, "", true);
        atts = new ArrayList(); // layer attributes
        for(i=1;i<fs.attributeCount;i++)atts.add(fs.getAttributeName(i));
    mid.addComboBox("Choose an attribute", null, atts, "");
    mid.pack();
    mid.show();
    if(!mid.wasOKPressed()) return;
    attribute = mid.getText("Choose an attribute");

    // map attribute values to list of geometries
    map = new HashMap();
    int count = 0;
    for (it = fc.iterator(); it.hasNext() ; ) {
        f = it.next();
        val = f.getAttribute(attribute);
```

```
            list = map.get(val);
            if (list==null) {
                list = new ArrayList();
                map.put(val, list);
            }
            list.add(f.geometry);
            // monitor has been set in the interpreter
            // in the addJumpPlugIn method to report progress
            Thread.sleep(1);
            monitor.report("Add feature " + (count++) + "/" + fc.size());
        }

        // Iterate over each individual attribute value
        it = map.keySet().iterator();
        while (it.hasNext()) {
            val = it.next();
            // monitor has been set in the interpreter
            // in the addJumpPlugIn method to report progress
            monitor.report("Processing " + val);
            merger = new LineMerger();
            merger.add(map.get(val));
            geoms = merger.getMergedLineStrings();
            it2 = geoms.iterator();
            while (it2.hasNext()) {
                newFeature = new BasicFeature(fc.featureSchema);
                newFeature.setGeometry(it2.next());
                newFeature.setAttribute(attribute, val);
                ds.add(newFeature);
            }
        }
        wc.layerManager.addLayer("Result",layers[0].name+"_merged",ds);
    }
    lineMerging();
```

Enregistrer ce script dans un fichier nommé "LineMerging.bsh" et placé dans le répertoire "MyScripts" (le chemin relatif du script va définir la structure du menu).

Puis, à partir de l'éditeur BeanShell, lancer le script permettant de créer les PlugIns (le premier script), et immédiatement après, entrez dans la ligne de commandes l'instruction suivante :

```
addPlugIn("MyScripts/LineMerging");
```

Cette dernière opération va créer un nouvel élément de menu LineMerging dans dans un (nouveau) menu MyScripts. Un click sur cet élément de menu devrait lancer le script lineMerging, si une couche est sélectionnée.

Maintenant, votre script est lié à l'élément de menu du même nom. Si vous voulez modifier le comportement du PlugIn, il vous suffit d'ouvrir le script (lineMerging, celui qui fait le travail) et de le corriger, de le modifier jusqu'à ce qu'il accomplisse la tâche attendue.

Et pour retirer l'élément de menu, hmmm..., si vous avez une solution simple et élégante, faites la moi parvenir, et je l'ajouterai dans ma prochaine documentation !

A. Version history

- **Version 0.1.0** (13 nov. 2004)

- Première version publique