BeanShellEditor for JUMP

version 0.1.0

Résumé

BeanShellEditor is a free editor for beanshell scripts. Jump is an open source GIS. This article explains how to use BeanShellEditor as a JUMP PlugIn.

Table of Contents

How to install BeanShellEditor Plugin

What do you need ?

To use BeanShellEditor as a JUMP PlugIn, you need :

- A Java Virtual Machine 1.4+ installed on your system. You can download sun's JVM from java [http://java.com/en/].
- JUMP : the main project is at http://www.jump-project.org/ and several interesting branches may be found through the world :
  - JPP [http://jump-pilot.sourceforge.net/]
  - SIGLE [http://www.projet-sigle.org/]
  - AGILES [http://www.agiles.org/]
- BeanShellEditor : you should find BeanShellEditor for JUMP with this documentation. Otherwise, check on SIGLE [http://www.projet-sigle.org/] site.
- If you did not find them in the distribution you can grab beanshell and buoy packages at

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

Your installation directory

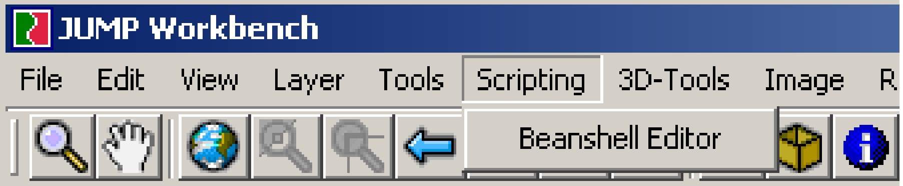
Your installation directory should look like that :

```
+ 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

To take full advantage of beanshell, you must download/install the last bsh-2.0b2 package (the package included in the 1.1.2 JUMP distribution is the bsh-2.0b1) .

Jump loads all the plugins located in the ext directory. The bsheditor plugin will create a new menu :



First steps with BeanShellEditor

In this section, we'll discover the use of JUMP API through simple beanshell scripts.

First, open the BeanShellEditor with the menu Scripting / BeanShell Editor.

All the objects contained in your application can be accessed through an object referenced as "wc", which is the WorkbenchContext

Tip

You can copy/paste the code samples of this document, run them, and save to your bsh directory the scripts you want to use or to modify later.

.

Printing information about layers

Print the complete list of layers

In the editor, write the following script :

```
// get a list of layers from the layer manager
layers = wc.getLayerManager().getLayers();
// for each layer, print the name of the layer
for(i=0 ; i<layers.size() ; i++) print(layers.get(i));
```

Then press the RUN button above the editor panel, and you'll get the list of layers in the output window of BeanShellEditor.

With beanshell 2.0, you can use a simplified syntax for accessors and loops :

```
// get a list of layers from the layer manager
layers = wc.layerManager.layers;
// for each layer in layers, print the name of the layer
for(layer:layers) print(layer);
```

You can even print an array or a collection in a one line statement :

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

An array/collection is printed by beanshell as a list of comma separated values between two brackets :

```
[COUNTRY, CITY, RIVER, ROAD]
```

Print the schema of selected layers

To print the schema associated with each layer, you should write :

```
//**************************************************
// PRINT THE SCHEMAS OF SELECTED LAYERS
//**************************************************
// get the selected layers from the LayerNamePanel
layers = wc.getLayerNamePanel().getSelectedLayers();
// for each layer, print the name of the layer and get the shema
// note that here, layers is an array (see JUMP API)
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

In java 1.4, you must use different syntaxes for an array and for a list :

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

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

`With java 5.0 and beanshell 2.0, you can use the enhanced style for`

```
both kind of loop :

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

Print the number of features of each layer in an HTML frame

To create a new HTML document in JUMP output window, just write :

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

htmlFrame.createNewDocument();
```

You can write in this frame, using usual html tags :

```
//************************************************
// OUTPUT LAYER'S NAME AND THEIR NUMBER OF FEATURES
// TO THE JUMP OUTPUT (HTML) FRAME
//************************************************
// get the JUMP output html frame
htmlFrame = wc.getWorkbench().getFrame().getOutputFrame();
htmlFrame.createNewDocument();

// initialize the 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>");

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

// Brings the output window to the front
htmlFrame.surface();
```

Working with layers

Creating a new empty layer from an existing one

Here, you'll create a new empty layer with a feature schema picked up from the selected layer :

```
//************************************************
// CREATE AN EMPTY LAYER
// FROM THE SCHEMA OF A SELECTED LAYER
// (first select a layer in an active task)
//************************************************

import com.vividsolutions.jump.feature.*;

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

// one layer must be selected
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));
}
```

If you create empty layers from all the selected layers, remove the second test, and include the code in a loop like :

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

Creating a layer with a specific feature schema

```
//**************************************************
// CREATE A LAYER WITH A CUSTOM SCHEMA
//**************************************************
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));
```

Creating a new layer from a selection

Here, you will select some features from a layer and copy them in a new layer. This script needs a specific layer to be selected and containing an numeric attribute called "Population".

Note

In §2.3 and § 3, you'll learn how to include code into generic methods or classes.

```
//****************************************
// CREATE A NEW LAYER FROM A SELECTION
// To run this script, you need a numerical
// "Population" attribute in the selected layer
//****************************************
import com.vividsolutions.jump.feature.*;

// Set the name of your selection (new layer)
selection = "Country50M";

// Get the selected layer
```

```
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);
        }
    }
}

// Create the new layer from the selection
// remark : features in the new layer are hard copies of selected features
wc.getLayerManager().addLayer("Working", selection, featureDataset);
```

Creating a layer for each type of geometry

Some data models need to have homogeneous types of geometry in their layers. Here is a script which will distribute different geometry types in different layers.

```
//**************************************
// CREATE A LAYER FOR EACH GEOMETRY TYPE
// A layer must be selected
// Even empty layers are created
//**************************************
import com.vividsolutions.jump.feature.*;

// Get the selected layer
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
    layer = layers[0];
    fc = layer.getFeatureCollectionWrapper();
    fs = fc.getFeatureSchema();

    // Create the layers derived from layer;
    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));

    // Test all the features of the collection with an iterator
    for (it = fc.iterator() ; it.hasNext() ; ) {
        feature = it.next();
        wc.getLayerManager()
          .getLayer(layer.getName()+"_"+feature.getGeometry().getGeometryType())
          .getFeatureCollectionWrapper()
          .add(feature);
    }
}
```

Creating new methods, classes and threads

Creating a method to calculate are of the selected features

Here, you will just write the method in the script editor, and use it as a convenien tool in the command line text area :

```
//*********************
// SELECTED ITEM'S AREA
//*********************
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;
}
```

Execute the script, then, try it in the command line text area on different feature selections :

```
area();
```

You can also use the variables & methods pane and click on the method name (#`area`) to write it in the command line text area. Don't forget to add a ";"

Creating a class representing a condition for selections

In this code, a condition is represented as a class used to execute a query :

Warning

To execute this code, you must have `bsh-2.0b2.jar` in the classpath instead of bsh-2.0b1.jar which is distributed with jump 1.1.2. You may have to modify your classpath in the .bat launcher too.

```
//**************************************************
// IMPLEMENTING A CONDITION INTERFACE TO EXECUTE
// A QUERY ON A LAYER
// This script does'nt execute any query but define
// a frame for query execution (see following script)
//**************************************************

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

// INTERFACE FOR A CONDITION
public interface Condition {public boolean test(Feature f);}


// QUERY METHOD USING A CONDTION INSTANCE
// RETURN AN OBJECT WITH 3 USEFUL METHODS
// - getResult() return the selected sataset
// - createNewLayer() create a layer from the selection
// - getAttributePanel() display the attribute panel
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;
}
```

And now, you can use it as follow (script or command line) :

```
// THIS EXAMPLE DISPLAY THE ATTRIBUTE TABLE CONTAINING
// THE SELECTION OF MyLayer FEATURES WITH LENGTH < 100
query(wc.layerManager.getLayer("MyLayer"),
    new Condition(){
        public boolean test(Feature f){return f.geometry.length<100;}
    }
).getAttributeTable();
```

You can also include the `Condition` and the `query` definition in a new script thanks to the beanshell `interpreter.source` command (see § 3.3).

Creating a Thread for long processes

See BeanShellEditor documentation

Advanced features

Here are some tips to customize your beanshell editor environment.

A directory for your scripts

Organize your script files in a directory. The directory name is, for example, bsh. Then, choose this directory with the script file folder option, and you'll get a one click access to all your scripts, through the script manager (on the left panel).

You can open your script (one click on the file name), modify it, and run it. If you want to save your changes, click on the save button (or save as).

A starting script to set a personal environment

Did you notice that some import statements, objects or method calls are repeatedly needed in your scripts. Let write a script which will always be run before you run a new script.

For example, your starting script should look like that :

```
//*********************************************************************
// Start beanshell script
// This script is interpreted each time a new interpreter is created
// by the beanshell editor (each time a script is executed from the
// editor panel)
// It is a good place for
//    - imports
//    - application attributes
//    - basic functions
//    - ...
//*********************************************************************

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

//*********************************************************************
// BASIC FUNCTIONS FOR PRINTING DATE OR FOR BENCHMARKING
//*********************************************************************

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 JUMP PACKAGES
//*********************************************************************

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.*;

//***************************************************************
// USEFUL HANDLES AND BASIC FUNCTIONS TO ACCESS JUMP LAYER
// wc is defined in the beanshell editor plugin for jump
//***************************************************************

// 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();
}


// Define a warning method
warning(String s) {wc.getWorkbench().getFrame().warnUser(s);}

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

Using several scripts together

A beanshell script may call another script (paths are relative to the directory from where you called your

JVM) :

```
// The path is relative to your application laucher
```

```
this.interpreter.source("../../bsh/jumptoolbox.bsh");
```

If you want to continue your program even if the file is not found, you must include it in a try/catch statement :

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

Practical example

In this example, we show how to separate the script containing reusable code (typically class and methods) from the script processing your data (typically, variable definition, and method execution).

The first one contains useful methods, while the second apply thoses methods to your own dataset.

```
//**********************************************
// ATTRIBUTE TOOLBOX CONTAINING
// - a method to trim() string attributes
// - a method to add an attribute issued from the
//   concatenation of two other attributes
//**********************************************
import com.vividsolutions.jump.feature.*;
import com.vividsolutions.jump.workbench.model.*;

// Removing leading and trailing white spaces from string attributes
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());
        }
    }
}

// Create a new String attribute defined by
// 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");
    }
}
```

Now, let's save the script somewhere (ex. `bsh/scriptutils.bsh`) and call it from another script specially written to process an ADDRESS layer (you must create this layer with a "NUMBER", and a "SREET" attribute to test this script) :

```
// remove the leading spaces and trailing spaces from my adress layer
layer = wc.layerManager.getLayer("ADDRESS");
for (f:layer.getFeatureCollectionWrapper().getFeatures()) trim(f);

// then add your concatened attribute
concat(layer, "NUMBER", "STREET", "ADDRESS", " ");
```

If you want a more robust script, don't hesitate to throw exceptions or to use the `wc.workbench.frame.warnUser()` method to warn the user that layer does not exist or that attribute names are not correct...

Using external libraries

In this example, we'll make use of itext pdf library to edit an atlas containg regional maps from the current task.

First statement include the new library in the classpath. If you put the library in the jump plugin directory, the statement should be :

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

```
//*****************************************************************
// CREATE A SIMPLE ATLAS FROM A JUMP TASK USING itext PDF LIBRARY
//*****************************************************************

// Adding itext library to the classpath
// the jar file is supposed to be in the current directory
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();
}
```

To apply this method to your map, call it like in the following command line :

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

Scripting new PlugIns

Here is a short example that demonstrate how to create a new threaded JUMP plugin calling a script. This is a very quick way to develop/debug/modify plugins.

Tip

Include the addPlugIn() function in your start script, and develop new PlugIns in only few minutes.

Warning

The BeanshellEditor has an addPlugin method which adds a plugin button to the editor.

```
//*****************************************************
// THIS METHOD CREATE A NEW PLUGIN FROM path.bsh FILE
//*****************************************************

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

addJumpPlugIn(String path) {
    final String plugInPath = path;
    // CREATE A CLASS IMPLEMENTING ThreadedPlugIn
    class BshPlugIn implements ThreadedPlugIn {
        // INITIALIZE A MENU-ITEM BASED ON path ARGUMENT
        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;
        }
        // RUN THE SCRIPT FROM 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;}
    }

    //CREATE A CLASS TO INITIALIZE THE PLUGIN;
    class MyExtension extends Extension {
        public void configure(PlugInContext context) throws Exception {
            new BshPlugIn().initialize(context);
        }
    }

    // CREATE AN INSTANCE OF THE EXTENSION
    new MyExtension().configure(wc.createPlugInContext());

}
```

And now, save this script and create the JUMP plugin doing the things (here, you want to create a PlugIn to merge lines of a layer, and call it through the menu item MyScripts --> lineMerging).

This script merge features of a layer containing lines when they touch each others and have a common attribute value (the value is choosen by the user from a combobox initilized with the values of the selected layer attributes) :

```
//***************************************
// 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 !");
    }
    // INITIALIZATION
    fc = layers[0].featureCollectionWrapper;
    fs = fc.featureSchema;
    ds = new FeatureDataset(fs);

    // INPUT DIALOG FOR ATTRIBUTE
    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();
```

Save this script in a file named "LineMerging.bsh" placed in a directory called "MyScripts" (the dir/file names will become the menu/menuitem names).

Then, from the BeanShellEditor, run the addPlugIn script (the first script), and enter the following one line command in the command line :

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

This will create a new menu item LineMerging in a MyScripts menu. A click on this menu item should launch your Isolation script, if a layer is selected.

Now, your script is linked to your menuitem. If you want to modify its behaviour, just open it, add your corrections, and call your menuitem again.

And to remove the menu item you just have created....hmmm, let me know how you did and I'll add the tip in next documentation !

A. Version history

- **Version 0.1.0** (13 nov. 2004)
  - First public version