Cytoscape.org

Plugin Developer Guide

Cytoscape.org

Plugin Developer Guide

About this Document

This document explains the basics of building Cytoscape plugins. It is primarily written for new Cytoscape developers.

Building Cytoscape Plugins: Overview

There are three main steps to building a Cytoscape plugin:

1. Create a class that extends the Cytoscape `AbstractPlugIn` class.
2. Create a Jar File for your plugin.
3. Obfuscate your plugin. You only need to do this step if you plan to release your plugin to the public.

Each of these steps is detailed below. To make everything concrete, a complete sample plugin is available within the main Cytoscape CVS server. To get started, make sure that you first do a complete checkout of the core Cytoscape code:

```
cvs checkout –P cytoscape
```

Then, do a complete checkout of the csplugins directory:

```
cvs checkout –P csplugins
```

The sample plugin that accompanies this document is available in csplugins/tutorial.

Step 1: Extend `AbstractPlugIn`

For the first step, you need to create a new class that extends the Cytoscape `AbstractPlugIn` class. The full source code for `AbstractPlugIn` is provided in Listing 1.

Listing 1: Source Code for the `AbstractPlugIn` class.

```
package cytoscape;

/**
 * AbstractPlugin is the class that all plugins must subclass;
 * the interface is simple - the constructor must take a single
 * {@link cytoscape.CytoscapeWindow CytoscapeWindow} argument,
 * and there must be a {@link #describe describe} method
 * returning a String description of the plugin.
 */
public abstract class AbstractPlugin {
    /**
     * this method's presence is superfluous;
     * it is only here so that you don't have to
     * call super(cytoscapeWindow) in your ctor.
     */
    public AbstractPlugin() { }
    /**
     * required constructor for plugins takes a single
     * {@link cytoscape.CytoscapeWindow CytoscapeWindow} argument.
     */
    public AbstractPlugin(CytoscapeWindow cytoscapeWindow) { }
    /**
     * method returning a String description of the plugin.
     */
    public String describe() { return new String("No description."); }
}
```

As you can see from the code above, new plugins are required to have two things:

1. First, your class must provide a constructor that receives a `CytoscapeWindow` object. The `CytoscapeWindow` is the main Cytoscape class, and provides access to the full set of Cytoscape menus and graph objects.
2. Second, your class must provide an implementation of the `describe()` method. This method should return a short description of what your plugin actually does.

By way of example, let's build a simple plugin that counts the number of nodes currently selected by the user. From a user perspective, here's how it works: the user selects zero or more nodes in the main Cytoscape window, and then selects menu PlugIns --> Count Number of Selected Nodes.

To build this simple plugin, we will create two classes. The first class, `NodeCounter` will extend `AbstractPlugIn`, and add a menu item to the plugin drop down menu. See Listing 2. The second class, `NodeCounterTask` will perform the actual node counting.

Listing 2: NodeCounter.java

```
package csplugins.tutorial;

import cytoscape.AbstractPlugin;
import cytoscape.CytoscapeWindow;

import javax.swing.*;

/**
 * Node Counter Plugin.
 * Illustrates the basics of building Cytoscape plugins.
 *
 * @author Ethan Cerami
 */
public class NodeCounter extends AbstractPlugin {

    /**
     * Constructor.
     * @param cWindow Main Cytoscape Window object.
     */
    public NodeCounter(CytoscapeWindow cWindow) {
        NodeCounterTask task = new NodeCounterTask (cWindow);
        JMenu menu = cWindow.getOperationsMenu();
        JMenuItem item = new JMenuItem("Count Number of Selected Nodes");
        item.addActionListener(task);
        menu.add(item);
    }

    /**
     * Describes the plugin.
     * @return short plugin description.
     */
    public String describe() {
        return new String ("Counts number of selected nodes");
    }
}
```

As required by the `AbstractPlugin` class, our class provides a constructor that receives a `CytoscapeWindow` object, and provides an implementation of the `describe()` method. Also, note that you can add new items to the Cytoscape plugin menu via the `getOperationsMenu()` method. In this case, we add a single new item to the plugin menu, and specify that the `NodeCounterTask` will receive these events.

The source code for the `NodeCounterTask` is provided in Listing 3.

Listing 3: NodeCounterTask.java

```
package csplugins.tutorial;

import cytoscape.CytoscapeWindow;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import y.view.Graph2D;
import y.base.NodeCursor;

import javax.swing.*;
```

```
/**
 * Counts Number of Selected Nodes.
 * Displays Number of Nodes within a simple Dialog Box.
 *
 * @author Ethan Cerami
 */
public class NodeCounterTask implements ActionListener {
    private CytoscapeWindow cWindow;

    /**
     * Constructor.
     * @param cWindow Main Cytoscape Window Object.
     */
    public NodeCounterTask (CytoscapeWindow cWindow) {
        this.cWindow = cWindow;
    }

    /**
     * Receives Menu item select event.
     * @param event Menu Item Selected.
     */
    public void actionPerformed(ActionEvent event) {
        //  Get Graph from Cytoscape Window
        Graph2D graph = cWindow.getGraph();
        //  Get All Selected Nodes and count them
        NodeCursor nodeCursor = graph.selectedNodes();
        int numNodesSelected = nodeCursor.size();
        //  Display Simple Dialog Box
        JOptionPane.showMessageDialog (cWindow, "Number of selected nodes:  "
                +numNodesSelected);
    }
}
```

This class waits for the user to select the "Count Number of Nodes" item from the plugin menu. When the user selects the menu item, `actionPerformed()` is called, and we use the `Graph2D` object to determine the total number of selected nodes.

Step 2: Create a Jar File

Cytoscape now has the ability to load plugins directly for Jar files. In order to use this functionality, you need to package your plugin into one Jar file. You can do this via the command line, via a makefile, or via an Ant build file. To help get you started, the csplugins/tutorial directory contains a sample Ant build file that you can use as a template.

Here's how it works. First, cd to the correct directory:

```
cd  csplugins/tutorial
```

then type

```
ant  jar
```

This will compile the sample plugin, and create a new Jar file: tutorial.jar. The Jar file will now be located in csplugins/tutorial/build.

Now, you need to verify that the plugin Jar file actually works. To do this, you need to run Cytoscape, and load your plugin from the File menu. Here's how it works. First, cd to the main cytoscape directory:

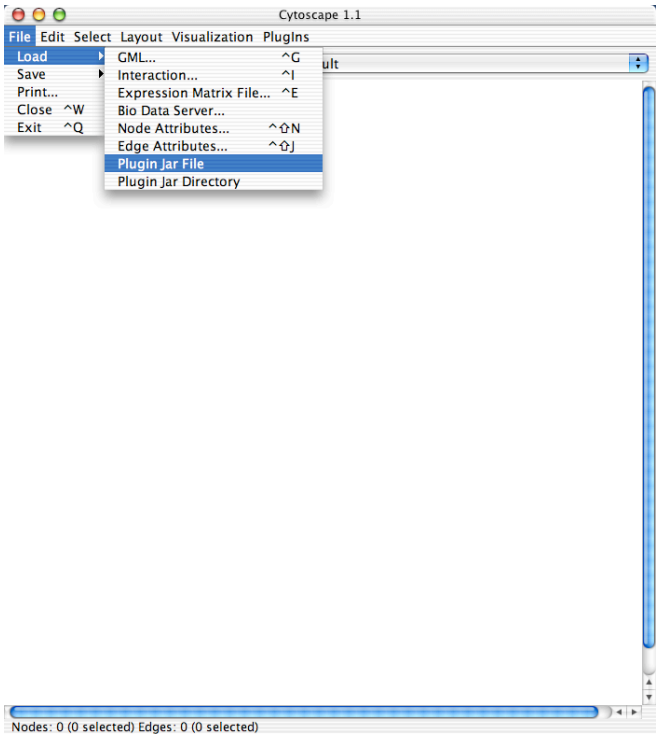
```
cd cytoscape
```

then type:

```
ant run
```

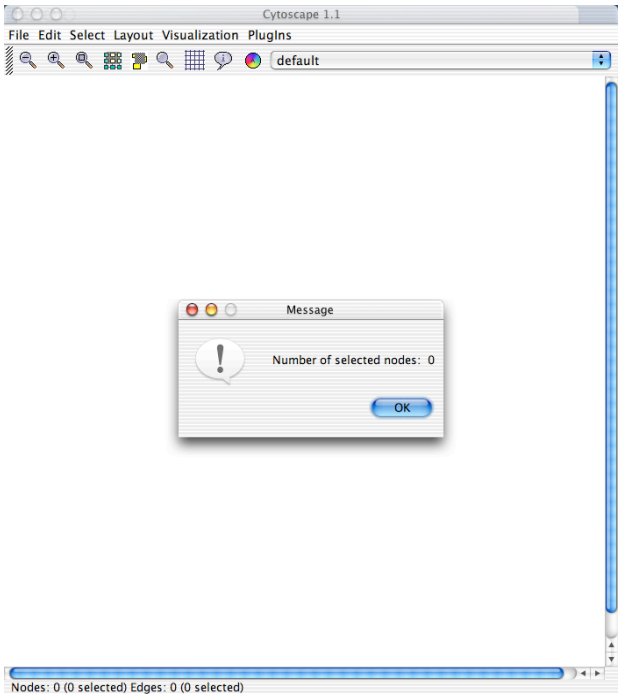
You should now see the main Cytoscape application on your desktop. To load the plugin, select File → Load → Plugin Jar File. See Figure 1. Then, specify the tutorial.jar file. Again, it is located in csplugins/tutorial/build.

Figure 1: Loading the Plugin Jar File.



Once the plugin has been loaded, click the PlugIns menu item. You should now see an item, labeled "Count Number of Selected Nodes." Select this item, and you should see something like that shown in Figure 2. If you see this, everything is working correctly.

Figure 2: The Node Counter In Action.



Step 3: Obfuscate Your Plugin

If you want to release your plugin to the public, your must obfuscate your plugin Jar file. This is required by the Y-Files license agreement. Once we move away from Y-Files, this final step will no longer be necessary.

As in Step 2, the sample Ant build file can automate this entire process for you. However, a little background will help understand how it works. First off, we currently use an open source tool, called Retroguard that obfuscates our code. Complete details on Retroguard are available at: *http://www.retrologic.com/retroguard-main.html*. The FAQ is available here: *http://www.retrologic.com/retroguard-faq.html*.

Second, each time Retroguard obfuscates a set of classes, it generates an *obfuscation table*. For example, it may map the class: y.layout.GraphLayout to something like: a.bb.c. In order to run a Cytoscape application with a Cytoscape plugin, both sets of Jar files must be obfuscated with the same obfuscation table. Otherwise, you will get lots of ClassNotFoundExceptions. The current Cytoscape obfuscation table is currently checked into CVS under: cytoscape/tools/jars/obfuscation_table.log.

Unfortunately, the current process of manually obfuscating a plugin is rather tedious and very error prone. You should therefore use the sample Ant build file as a template. Here's how it works.

1. First, you must obfuscate the Cytoscape core. To do so, cd to the main cytoscape directory:

cd cytoscape

then, type:

ant obfuscate

This will run Retroguard on Cytoscape and Y-Files with the current obfuscation table. It will then create a file called, cytoscape-obfuscated.jar, which will be located in the `cytoscape/dist` directory.

2. Next, you need to obfuscate your plugin. To do so, cd to the correct directory:

   cd csplugins/tutorial

   Then, type:

   ant obfuscate

   This will run Retroguard on your plugin with the current obfuscation table. It will then create a file, called tutorial-obfuscated.jar, which will be located in the `csplugins/tutorial/dist` directory.

3. Finally, you need to verify that everything works. If you type: ant run, this will run Cytoscape with the regular non-obfuscated Jar file. To run Cytoscape with the obfuscated Jar file, first go back to the regular cytoscape directory:

   cd cytoscape

   then, type:

   ant run_obfuscated

   Now try loading the obfuscated plugin from the Menu: File → Load → Plugin Jar File. Verify that you can actually use the plugin. If you are able to see something like that shown in Figure 2, and do not get any ClassNotFoundExceptions, everything is set up correctly.

Working with the Ant Build Template

The build.xml file currently checked into csplugins/tutorial is designed to serve as a template for other plugin developers. Of course, you are free to use whatever build process you like, but this template should get you started quickly.

To understand the build.xml file, you will need to understand the basics of Ant. If you do not have this background, check out the Ant Manual online at: *http://ant.apache.org/manual/*. If you don't want to bother with understanding Ant, email Ethan, and he can adapt the template to your specific plugin.

The complete build.xml file is shown in Listing 4. Details on the most important parts follow.

Listing 4: Template Build.xml File

```
<?xml version="1.0"?>
<!-- build.xml - Build file for Sample Cytoscape Plugin -->
<project name="Cytoscape" default="jar" basedir=".">

  <!-- Name of Plugin -->
  <!-- Change this to the name of your Plugin -->
  <!-- The name is used to create the plugin JAR file -->
  <property name="plugin.name" value="tutorial"/>

  <!-- The directory containing source code -->
  <property name="src.dir" value="src"/>

  <!--  The directory containing library jar files -->
  <property name="lib.dir" value="../../cytoscape/lib"/>

  <!-- Temporary build directories -->
  <property name="build.dir" value="build"/>
  <property name="build.classes" value="${build.dir}/classes"/>
  <property name="dist.dir" value="dist"/>

  <!-- Global "magic" property for <javac> -->
  <property name="build.compiler" value="modern"/>

  <!-- Classpath with all lib JAR Files and all Cytoscape Core classes -->
  <path id="classpath">
    <pathelement path="${build.classes}"/>
    <pathelement path="../../cytoscape/build/classes"/>
    <fileset dir="${lib.dir}">
      <include name="**/*.jar"/>
    </fileset>
  </path>

  <!-- Target to create the build directories prior to compilation -->
  <target name="prepare">
       <mkdir dir="${build.dir}"/>
       <mkdir dir="${build.classes}"/>
    <mkdir dir="${dist.dir}"/>
  </target>

  <!--  Target to clean out all directories -->
  <target name="clean" description="Removes all generated files.">
       <delete dir="${build.dir}"/>
    <delete dir="${dist.dir}"/>
  </target>

  <!-- Target to create Cytoscape Jar File  -->
  <target name="build_cytoscape">
    <ant antfile="../../cytoscape/build.xml" inheritAll="false" target="jar"/>
  </target>

  <!--  Target to compile all Plugin Code -->
  <target name="compile" depends="prepare, build_cytoscape"
    description="Compiles all PlugIn source code.">
       <javac srcdir="${src.dir}" destdir="${build.classes}" nowarn="on" source="1.4">
      <include name="csplugins/**"/>
      <classpath refid="classpath"/>
    </javac>
  </target>

  <!--  Create PlugIn Jar File (Unobfuscated) -->
  <target name="jar" depends="compile" description="Creates PlugIn Jar File.">
    <jar destfile="${build.dir}/${plugin.name}.jar">
      <fileset dir="${build.classes}"/>
    </jar>
    <echo message="${plugin.name}.jar is now complete.  It is located in build/
```

```
      directory"/>
  </target>

  <!-- Obfuscate the Plugin Jar File -->
  <target name="obfuscate" depends="jar" description="Obfuscates PlugIn Jar File.">
    <!-- Obfuscate Cytoscape Core -->
    <ant antfile="../../cytoscape/build.xml" inheritAll="false" target="obfuscate"/>

    <!-- Copy cytoscape.jar (with yfiles) to distribution directory -->
    <copy file="../../cytoscape/dist/cytoscape.jar" todir="${dist.dir}"/>

    <!-- Add PlugIn classes to cytoscape.jar -->
    <jar destfile="${dist.dir}/cytoscape.jar" update="true">
      <fileset dir="${build.classes}"/>
    </jar>

    <!-- Copy Obfuscation Table to distribution directory -->
    <copy file="../../cytoscape/tools/jars/obfuscation_table.log" todir="${dist.dir}"/>

    <!-- Obfuscate Cytoscape Jar File with RetroGuard -->
    <java classname="RetroGuard"
        classpathref="classpath" fork="true" dir="${dist.dir}">
      <arg line="cytoscape.jar cytoscape-obfuscated.jar obfuscation_table.log retro.log"/>
    </java>

    <!-- Extract csplugin classes to new jar file -->
    <mkdir dir="${dist.dir}/temp"/>
    <unjar src="${dist.dir}/cytoscape-obfuscated.jar" dest="${dist.dir}/temp"/>
      <jar destfile="${dist.dir}/${plugin.name}-obfuscated.jar">
        <fileset dir="${dist.dir}/temp/">
          <include name="csplugins/**"/>
      </fileset>
    </jar>

    <delete file="${dist.dir}/cytoscape.jar"/>
    <delete file="${dist.dir}/cytoscape-obfuscated.jar"/>
    <delete file="${dist.dir}/obfuscation_table.log"/>
    <delete dir="${dist.dir}/temp"/>
      <echo message="${plugin.name}-obfuscated.jar is now complete.
        It is located in dist/ directory"/>
  </target>
</project>
```

The build.xml file has a number of targets. The first to consider is the “compile” target:

```
<!--  Target to compile all Plugin Code -->
<target name="compile" depends="prepare, build_cytoscape"
  description="Compiles all PlugIn source code.">
    <javac srcdir="${src.dir}" destdir="${build.classes}" nowarn="on" source="1.4">
      <include name="csplugins/**"/>
      <classpath refid="classpath"/>
    </javac>
</target>
```

This target will compile all .java files located in the `src.dir` directory. By default, all sources are assumed to be located in the `src` directory. If this does not match your plugin directory structure, simply update this line:

```
<property name="src.dir" value="src"/>
```

Change the *value* attribute to point to your source directory.

The next target to consider is the “jar” target:

```
<!--  Create PlugIn Jar File (Unobfuscated) -->
```

```
<target name="jar" depends="compile" description="Creates PlugIn Jar File.">
  <jar destfile="${build.dir}/${plugin.name}.jar">
    <fileset dir="${build.classes}"/>
  </jar>
  <echo message="${plugin.name}.jar is now complete.  It is located in build/ directory"/>
</target>
```

This target will create a Jar file with everything in the `build.classes` directory. It will create a Jar file with the name specified by the *plugin.name* property. You should update this property to reflect your plugin:

```
  <!-- Name of Plugin -->
  <!-- Change this to the name of your Plugin -->
  <!-- The name is used to create the plugin JAR file -->
  <property name="plugin.name" value="tutorial"/>
```

Finally, consider the "obfuscate" target. There is a lot going on here, and most of it has to do with the quirky nature of using Retroguard. My advice is to simply copy over this target into your build file, and use it as is.