Plugin Development

Introduction

Savant is unique in the Genome Browser arena in that it was designed to be extensible through a rich plugin framework, which allows developers to provide functionality in addition to what is provided with the standard distrobution.

Tutorial: Developing a Savant Plugin

In this tutorial, we describe the steps used to create the DNARadio plugin using the demo project included in the Savant SDK as a template. The DNARadio plugin was inspired by an interesting project that is streaming the human genome one nucleotide at a time. Admittedly it's not a very useful plugin, but it's a fun example of some of the aspects of the API. Most importantly, **this tutorial was designed to enable you to create a foundation for your own plugin project**.

Prerequisites:

- JDK
- Netbeans
  - Note: for this tutorial, it is recommended to download the Netbeans + JDK Bundle
- Savant SDK

A. Set up the Netbeans Project

1. Set up Netbeans and a the JDK

Download Netbeams and the JDK and install them. The links for these are listed above. For simplicity it is recommended to download the bundle, but you can certainly download and install them independently if you wish.

2. Set up the Savant SDK

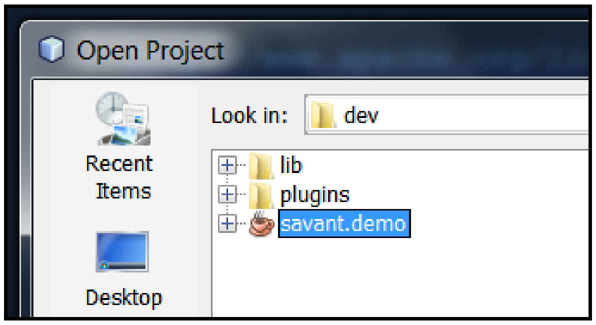
Download the Savant SDK, listed above, and unzip it. The SDK contains the following components:

- api - documentation of the Savant code
- dev - development materials
- development manual

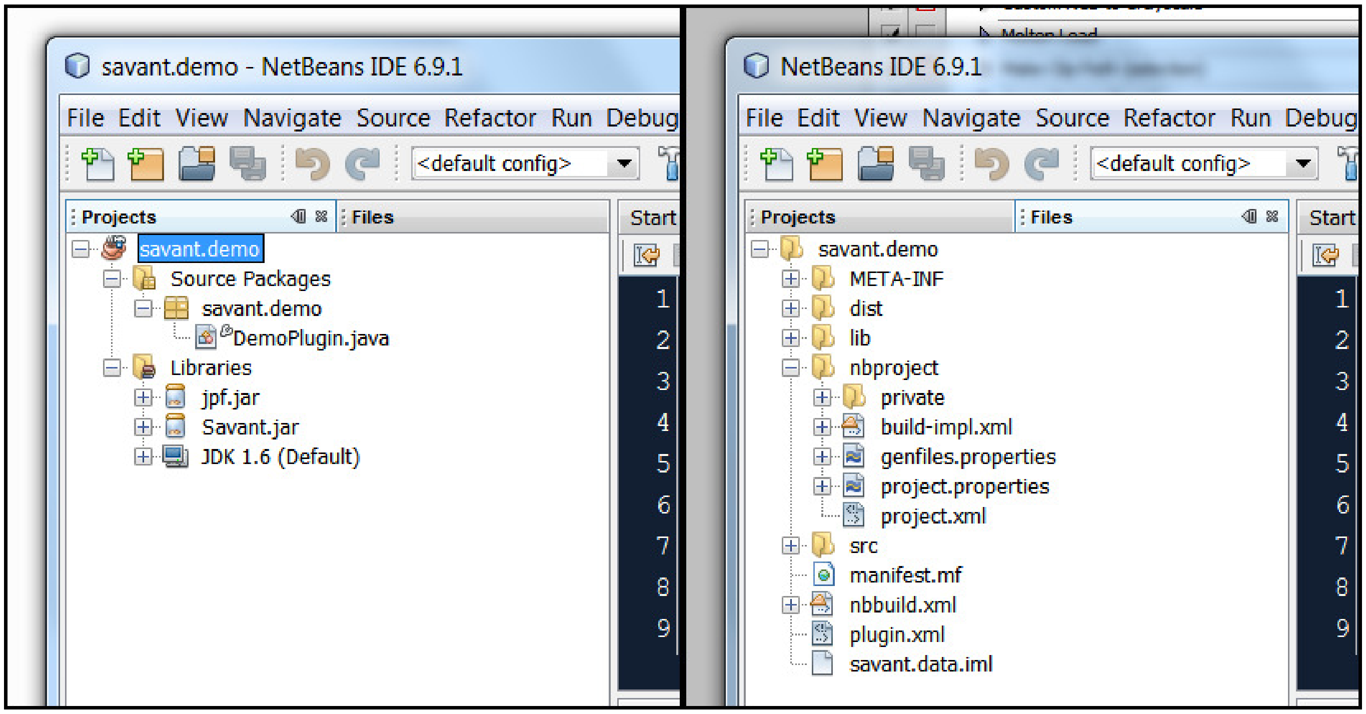
For this tutorial, we will refer to the dev folder in the SDK as dev/.

3. Open the Demo Project

Open Netbeans. Open the project in dev/savant.demo.



The "Project" and "Files" views should look something like this:



Description of Important Project Files:

- Project View
  - Source Packages: contains implementation of this plugin
    - Main package (savant.demo): contains the main class of the plugin
    - Main class (DemoPlugin.java): the class which implements the Savant Plugin interface
  - Libraries: contains links to external code packages
    - jpf.jar: Java Plugin Framework, required for loading plugins
    - Savant.jar: Savant API, required for accessing Savant functions
- Files View
  - lib/: contains additional libraries used by this plugin
  - nbproject/project.properties : contains Netbeans project properties
  - nbbuild.xml: contains Ant instructions for building the project
  - plugin.xml: contains information about the plugin and how to run it

B. Editing the Demo Project to Your Own Plugin

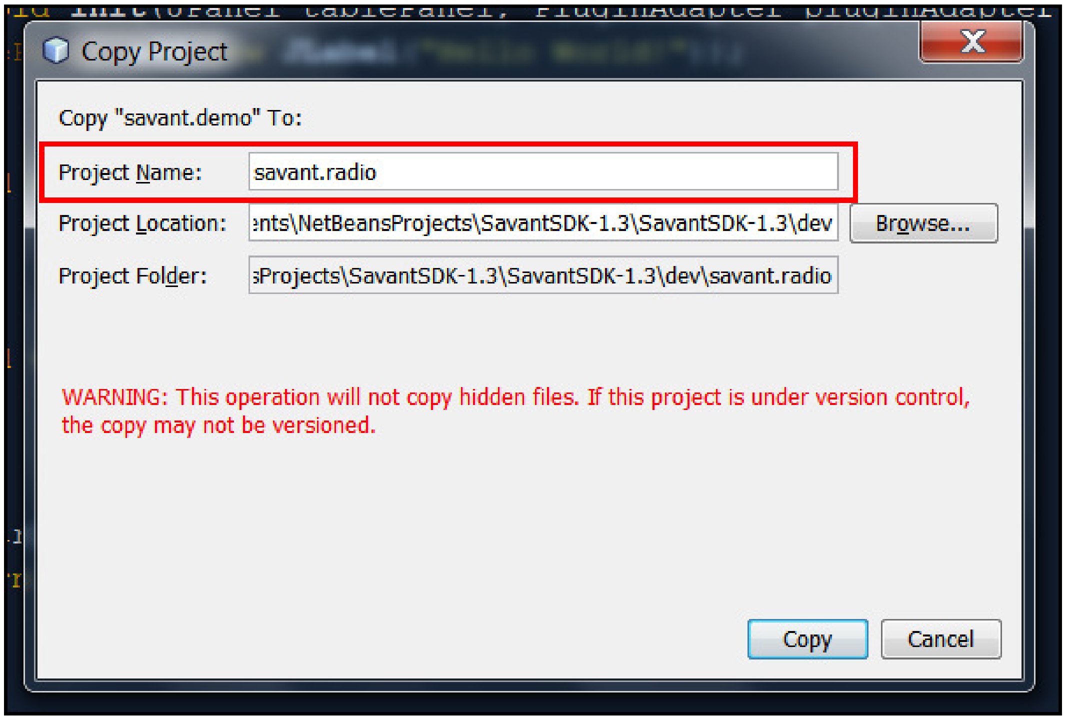
You will need to know the following properties of your plugin:

- NAME: the name of the plugin.
- VERSION: the version of the plugin. In Savant we use the following scheme for version numbers [major version] . [minor version] . [bug fix / build number].
- PACKAGE: the main package of the plugin (which contains the main class).
- CLASS: the main class of the plugin.

In this example, NAME="DNA Radio", VERSION="1.0.0", PACKAGE="savant.radio", and CLASS="DNARadio".

1. Copy the Demo Project

Right-click the savant.demo project icon (looks like a coffee cup) and choose "Copy...". The copy dialog will be presented. In the Poject Name field, enter PACKAGE. Click the "Copy" button.


Copy Project

Copy "savant.demo" To:

Project Name: savant.radio

Project Location: ents\NetBeansProjects\SavantSDK-1.3\SavantSDK-1.3\dev

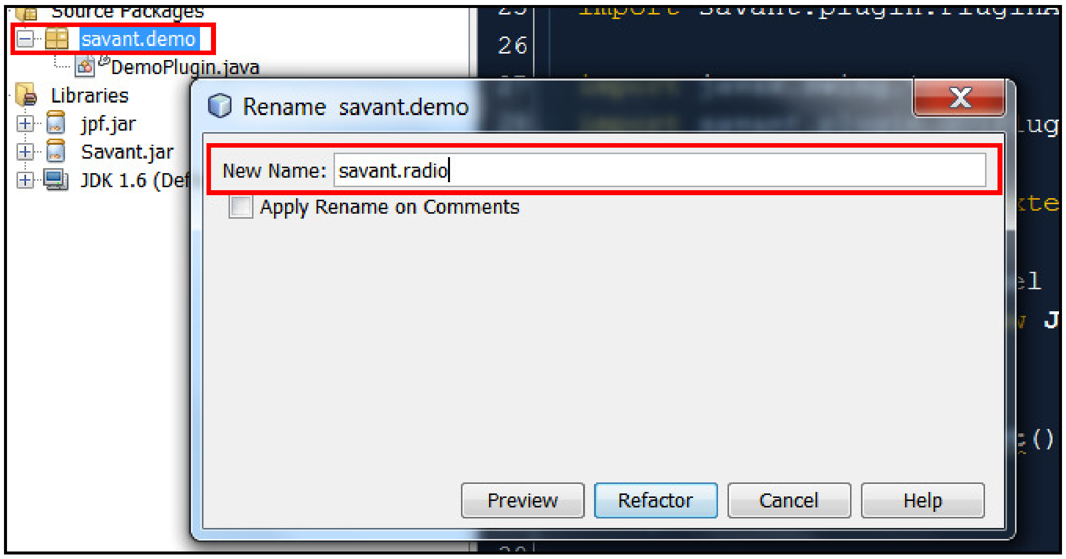
Project Folder: sProjects\SavantSDK-1.3\SavantSDK-1.3\dev\savant.radio

WARNING: This operation will not copy hidden files. If this project is under version control, the copy may not be versioned.

This will create a copy of the demo project which we will now edit to create a new plugin.

2. Rename the main package

Right-click the savant.demo package icon (looks like a cardboard box) and choose "Refactor > Rename...". In the New Name field, enter PACKAGE. Click the "Refactor" button.


Rename savant.demo

New Name: savant.radio

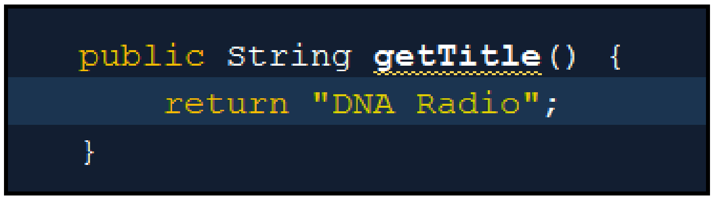
Apply Rename on Comments

3. Rename the main class

Right-click the DemoPlugin.java icon and choose "Refactor > Rename...". In the New Name field, enter CLASS. Click the "Refactor" button.

4. Change the plugin name

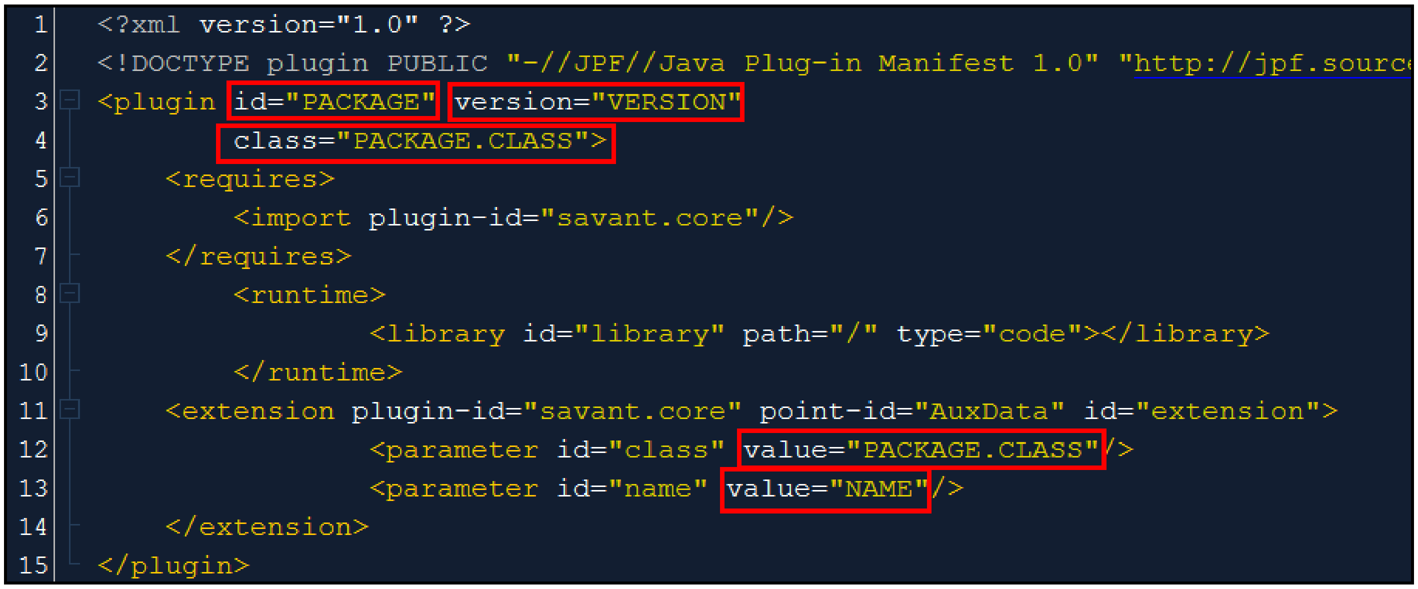
Double-click the CLASS.java file (formerly DemoPlugin.java) to open it in the Editor. Change the getTitle function to return "NAME".



```java
public String getTitle() {
    return "DNA Radio";
}
```

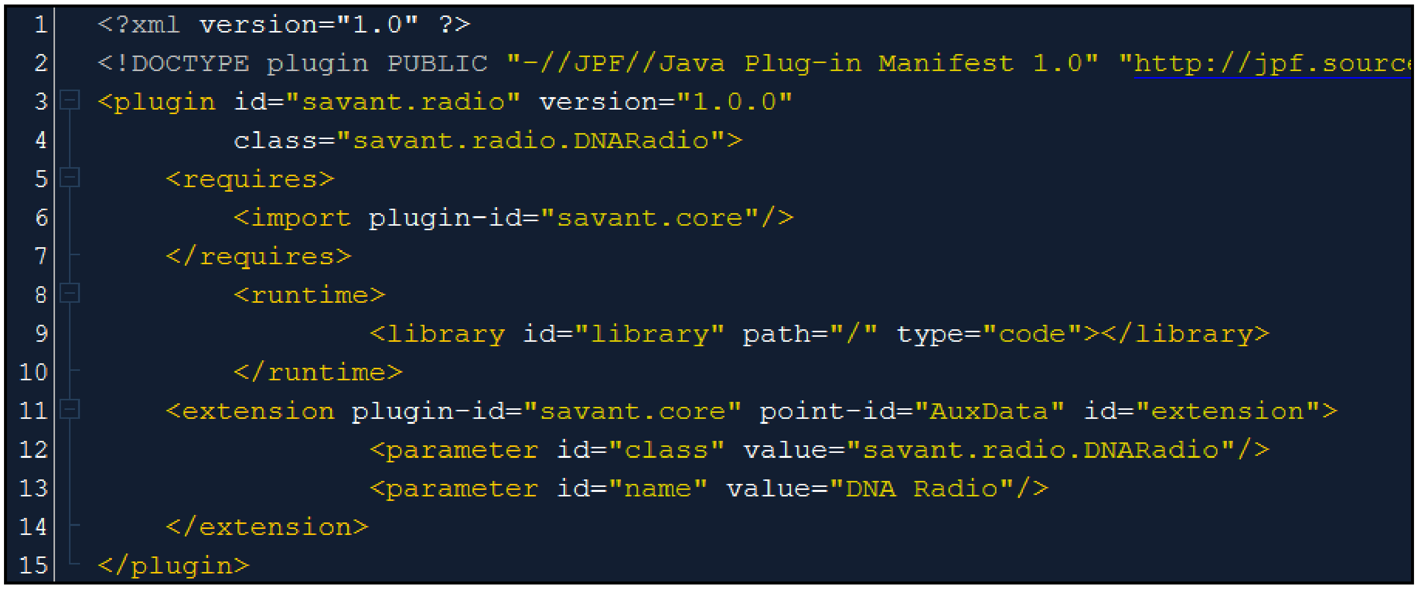
5. Edit the plugin properties

Switch to the "Files" view (the tab beside the "Projects" tab). Expand the new project. Double-click the plugin.xml file to open it in the editor. There are 5 places that need to be edited in this file, denoted in red boxes in the following figure:



```xml
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0" "http://jpf.sourc
<plugin id="PACKAGE" version="VERSION"
        class="PACKAGE.CLASS">
    <requires>
        <import plugin-id="savant.core"/>
    </requires>
        <runtime>
                <library id="library" path="/" type="code"></library>
        </runtime>
    <extension plugin-id="savant.core" point-id="AuxData" id="extension">
                <parameter id="class" value="PACKAGE.CLASS"/>
                <parameter id="name" value="NAME"/>
    </extension>
</plugin>
```

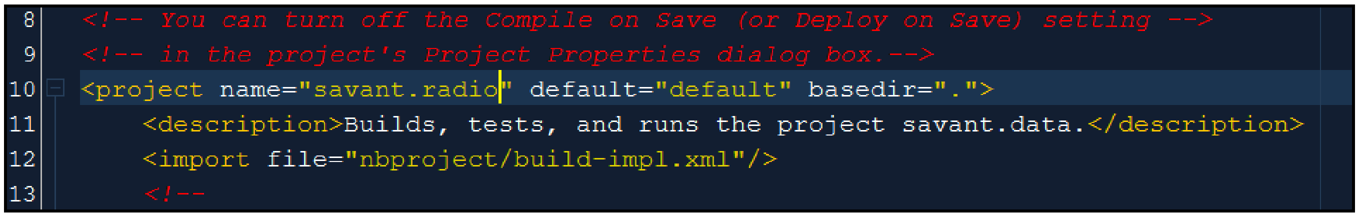
Here is how the file will be changed in the DNARadio example:



```xml
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 1.0" "http://jpf.sourc
<plugin id="savant.radio" version="1.0.0"
        class="savant.radio.DNARadio">
    <requires>
        <import plugin-id="savant.core"/>
    </requires>
        <runtime>
                <library id="library" path="/" type="code"></library>
        </runtime>
    <extension plugin-id="savant.core" point-id="AuxData" id="extension">
                <parameter id="class" value="savant.radio.DNARadio"/>
                <parameter id="name" value="DNA Radio"/>
    </extension>
</plugin>
```

6. Change the project name

Open the nbbuild.xml file in the editor (it's in the same directory as plugin.xml). Change the name attribute of the project to PACKAGE.



```
<!-- You can turn off the Compile on Save (or Deploy on Save) setting -->
<!-- in the project's Project Properties dialog box.-->
<project name="savant.radio" default="default" basedir=".">
    <description>Builds, tests, and runs the project savant.data.</description>
    <import file="nbproject/build-impl.xml"/>
    <!--
```

At this point, we have created an independent plugin project which can be built and loaded into Savant. It is a good idea to try to do so now. See the section on "Building Plugins" section below.

C. Code Away!

You now have an independent plugin project as a foundation for your own plugin. You have two main tasks, outlined below. If you are following the DNARadio tutorial, you should download the complete DNARadio Netbeans Project to a separate location and copy the Java files in the dev/src directory of that package to the dev/src directory of your Netbeans project (overwriting DNARadio.java). Also, see the "Including Other Libraries" section below to include the jl1.0.1.jar, which contains a library for playing audio files.

In general, the remaining tasks are:

1. Implementing the init() function

The init function is called immediately when the plugin is loaded as Savant starts. The init function is responsible for initializing (among other things) the graphical components of the plugin (e.g. buttons, menus, etc.). The graphical "canvas" for plugins is a JPanel provided as an argument to the init function. For help with JPanel and other Swing components visit here.

2. Implementing the rest of your plugin

Go nuts, you are free to do whatever you like in your plugin!

Suggested guidelines:

- *thread long running tasks:* Your plugin will run in the main Savant thread unless you tell it to do otherwise. This means that if your plugin does something which takes a long time (e.g. downloads files, does a long computation, etc.) it will lock up the main thread until it's finished. Please do such tasks in a separate thread. See the tutorial on Concurrency for more information.
- *use only what's provided in the Savant Plugin Adapter*: the Plugin Adapter contains functions which the Savant Development Team considers useful for developers to have. It is possible a developer requires some more functionality that can be found by getting access to a Savant instance. Please notify us of these situations so that we can consciously include such functionality in subsequent versions of the API, otherwise there is a risk that this functionality will be destroyed (since we're not expecting anyone to be using it directly) and your plugin will no longer work.

D. Submit your plugin

We strongly encourage you to submit all your developed Savant plugins to us, so that we can make them directly available to all Savant users. This is both a great way to promote your own work while encouraging more users and developers to join the Savant Community! You can submit your plugins to us through our Plugin Submission Form.

Plugin Development Tips and Tricks

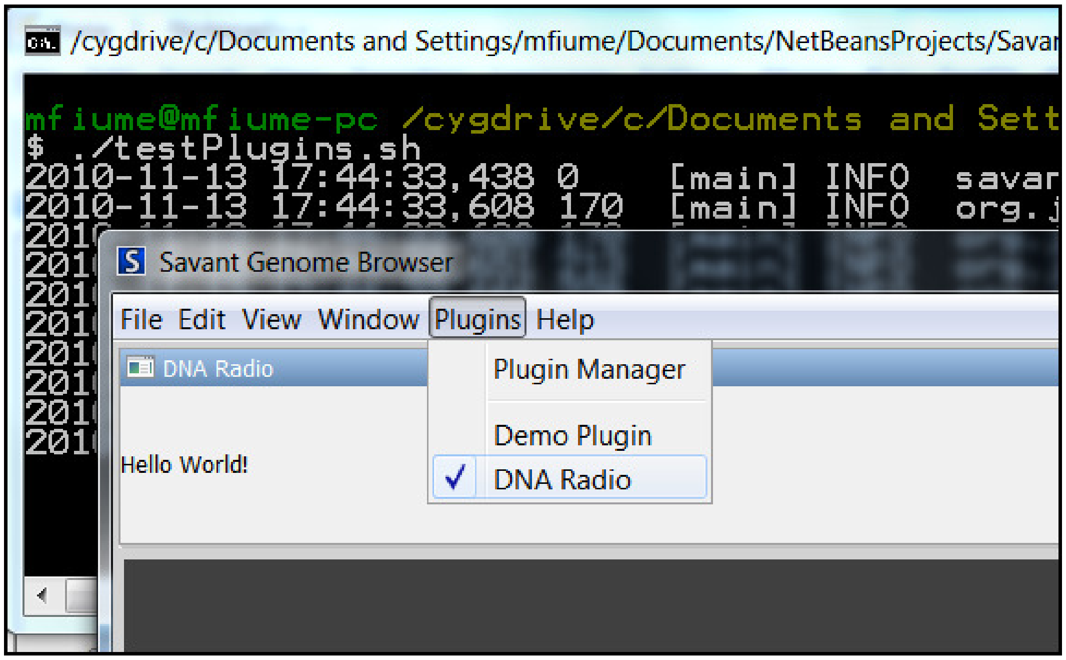
Building and Testing Plugins

To build your plugin, Right-click the plugin project icon (the coffee cup) and choose "Build". Netbeans should output something like this:

```
init:
deps-jar:
compile:
Copy libraries to C:\Users\mfiume\Documents\NetBeansProjects\Sav
Deleting directory C:\Users\mfiume\Documents\NetBeansProjects\Sa
Moving 1 file to C:\Users\mfiume\Documents\NetBeansProjects\Sava
BUILD SUCCESSFUL (total time: 0 seconds)
```

What's important is that it builds successfully. Look in the dev/plugins directory. You should see a .jar file corresponding to your plugin (in the DNARadio example, it is savant.radio-1.0.0.jar). Make sure that there is also SavantCore.jar in that directory. If you accidentally do a "Clean and Build" instead of a "Build", it may remove this file, which is required to load all plugins. If it's not there, you can always get a fresh copy from the SDK in the dev/plugin directory.

Now that you've ensured that the plugin was built, you can test out the plugin by running the provided dev/testPlugins.sh script or by running the Savant.jar otherwise (e.g. in Windows, you can just double-click Savant.jar).



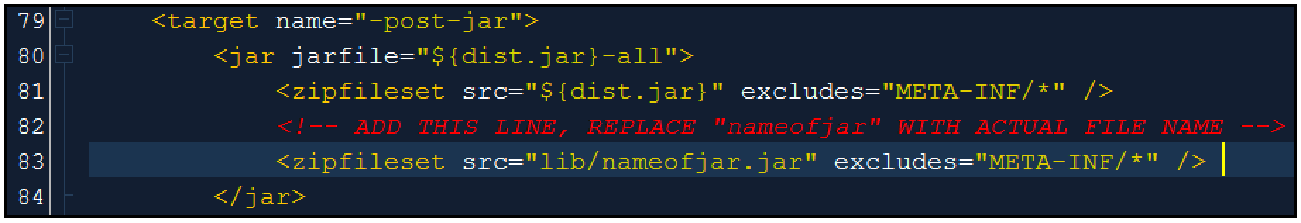
```
/cygdrive/c/Documents and Settings/mfiume/Documents/NetBeansProjects/Savar

mfiume@mfiume-pc /cygdrive/c/Documents and Sett
$ ./testPlugins.sh
2010-11-13 17:44:33,438 0    [main] INFO  savar
2010-11-13 17:44:33,608 170  [main] INFO  org.j
201
201
201
201
201
201
201
201
```

Including Other Libraries

You may include additional libraries in your plugin. To do so:

1. Place the jar file for the external library in the dev/lib folder.
2. In your Netbeans project, right-click "Libraries", then choose "Add JAR/Folder...", then find the jar file and click "Open".

3. Open the nbbuild.xml file in the "Files" view in Netbeans, and add a new zipfileset entry under the previous zipfileset entry (near the bottom of the file) for the jar file as follows:



```
    <target name="-post-jar">
        <jar jarfile="${dist.jar}-all">
            <zipfileset src="${dist.jar}" excludes="META-INF/*" />
            <!-- ADD THIS LINE, REPLACE "nameofjar" WITH ACTUAL FILE NAME -->
            <zipfileset src="lib/nameofjar.jar" excludes="META-INF/*" />
        </jar>
```

Troubleshooting

If you have any trouble developing a plugin, the Savant Development Team would be happy to help via email to savant [at] cs [dot] toronto [dot] edu.