CellDesigner^{TM} Plugin Tutorial

2014/06

The Systems Biology Institute
http://www.systems-biology.org/

Keio University, Dept. of Biosciences and Informatics,
http://fun.bio.keio.ac.jp/

Mitsui Knowledge Industry Co. Ltd.
http://bio.mki.co.jp/en/

TABLE OF CONTENTS

Acknowledgements:

- Rainer Machné and Christoph Flamm at University of Vienna for providing us a library version of SBML ODE Solver (SOSlib).
- Ralph Gauges, Sven Sahle and Ursula Kummer at University of Heidelberg for the development of a library version of COPASI.
- Frank Bergmann and Herbert Sauro at University of Washington for helping us support SBW-2.x on CellDesigner.
- Many thanks to the users who kindly provided us bug reports and feature requests!

CellDesigner^{TM} Plugin allows developers to extend the function of CellDesigner. This tutorial explains how to implement and load a plugin.

You can find the latest plugin information on the following website:
http://celldesigner.org/plugins.html.

1. Development Environment

To develop your CellDesigner Plugin, the following software packages are required or recommended.

(1) CellDesigner 4.4 (required)
(2) Java Development Kit (JDK) 1.6 or 1.7 (required)
(JDK 1.6 is recommended.)
(3) Eclipse (recommended)

2. The overview of the development of CellDesigner Plugin

(1) Put "celldesigner.jar" (located in "/path/to/celldesigner_installdir/exec/celldesigner.jar") and "sbmlj.jar" (located in "/path/to/celldesigner_installdir/lib/sbmlj.jar") in java class path.
(2) Implement your plugin java class. Please refer to the CellDesigner Plugin API documentation (located in "/path/to/celldesigner_installdir/documents/plugin/index.html")
(3) Archive the compiled plugin java class files as a java jar file.
(4) Deploy the jar file to the directory, "/path/to/celldesigner_installdir/plugin/".

3. CellDesigner Plugin API documentation.

Please refer to the CellDesigner Plugin API documentation,
which is stored in \document directory under your installed CellDesigner directory.

4. How to implement your plugin for CellDesigner

(1) Write your plugin class. Your plugin class must extend the CellDesignerPlugin class. CellDesigner will call the constructor of your plugin class to instantiate it.

```
public class SamplePlugin extends CellDesignerPlugin {
        /**
        * Constructor
        */
        public SamplePlugin(){

        }
}
```

(2) Write an action class which extends the PluginAction class for an action event that would be passed when the plugin menu is selected on CellDesigner. Implement a myActionPerformed method for the action event to realize your plugin functionality. The "setStarted" must be "true" in "myActionPerformed". The value means your plugin is active (invoked), and can call methods triggered by the events from CellDesigner (See 5.)

```
public class SampleAction extends PluginAction {

        public SampleAction(SamplePlugin _plugin){
                this.plugin = _plugin;
                // Write your codes for constructor.
        }

        public void myActionPerformed(ActionEvent e) {
                plugin.setStarted(true);
                // Write your codes for action event
```

(3) Use PluginMenu class and PluginMenuItem class to create menus on CellDesigner. Register the action class to the PluginMenuItem for CellDesigner to invoke the action.

```
public class SamplePlugin extends CellDesignerPlugin {
        /**
          * Constructor
          */
        public SamplePlugin(){
                PluginMenu menu        = new PluginMenu("Sample");
                SampleAction action = new SampleAction(this);
                PluginMenuItem item = new PluginMenuItem("sample1", action);
                menu.add(item);
                addCellDesignerPluginMenu(menu);
        }
}
```

(4) Use the following methods to register the PluginMenus to CellDesigner.

addCellDesignerPluginMenu; the method to register a menu to Plugin menu
addSpeciesPopupMenu; the method to register a menu to the popup menu which is invoked when a species is right clicked
addReactionPopupMenu; the method to register a menu to the popup menu which is invoked when a reaction is right clicked
addCompartmentPopupMenu; the method to register a menu to the popup menu which is invoked when a compartment is right clicked

(5) Implement the following methods for your plugin to receive events from CellDesigner. If not, the compilation will be failed.

SBaseAdded; this method is invoked when SBase is added to CellDesigner
SBaseChanged; this method is invoked when SBase is changed on CellDesigner
SBaseDeleted; this method is invoked when SBase is deleted from CellDesigner
modelOpened; this method is invoked when a model is opened or created on CellDesigner
modelSelectChanged; this method is invoked when the selected model is changed on CellDesigner
modelClosed; this method is invoked when a model is closed on CellDesigner

```
public class SamplePlugin extends CellDesignerPlugin {

        /**
         *
         */
        public SamplePlugin(){
                PluginMenu menu         = new PluginMenu("Sample");
                SampleAction action = new SampleAction(this);
                PluginMenuItem item = new PluginMenuItem("sample1", action);
                menu.add(item);
                addCellDesignerPluginMenu(menu);
        }

        public void addPluginMenu() {
        }

        public void SBaseAdded(PluginSBase sbase) {
        }

        public void SBaseChanged(PluginSBase sbase) {
        }

        public void SBaseDeleted(PluginSBase sbase) {
        }

        public void modelOpened(PluginSBase sbase) {
        }

        public void modelSelectChanged(PluginSBase sbase) {
        }

        public void modelClosed(PluginSBase sbase) {
        }

}
```

5. The information that your plugin can access

The information that your plugin can get from CellDesigner are the followings:

The selected model (SBML)
All models
A selected node on a model
All nodes on a model

6. Notification from Plugin to CellDesigner

You can implement the functions to add, update and delete PluginSBase in CellDesignerPlugin. The Plugin can notify CellDesigner of these changes via the CellDesignerPlugin interface. What parameter values you can modify is restricted, which conforms to the rules in CellDesigner (some parameter values can not be changed even in CellDesigner).

Example（in case of PluginSpecies)
Uneditable parameters : ID, Name and Type
Editable parameters : other parameters
If you need to edit Name and Type, you should edit PluginSpeciesAlias.

7. Restrictions

Some actions trigger sequential actions. For instance, once a species is deleted in CellDesigner, all reactions connected to the species are deleted. In this case, CellDesigner notify your plugin of the followings; 1) deleting species and 2) deleting reactions.

If species is deleted on your plugin, it notifies CellDesigner of the event. Then CellDesigner deletes the species and reactions connected to it. In this case, we recommend you to implement the function that delete the reactions associated with the deleted species in your Plugin because CellDesigner does not notify your Plugin of these deletions (CellDesigner does not notify Plugin of the event that is occurred by another event notified by Plugin in order to avoid forever loop).

Other examples
Update the compartment after moving a species
Update species including the compartment after updating compartment
Update the species’ compartment after updating the compartment
Update the species’ compartment after adding the compartment
Update name for Protein/RNA/Gene/asRNA

8. Sample Plugins

This section explains the details of a sample plugin. The following sample plugin can display name, ID, x position and y position of species selected on CellDesigner. If you click on “ADD” button, a new species is created and shown on CellDesigner.

8.1 Getting the properties of Species from CellDesigner

```
/**
 * get information on selected species
 *
 */
private void getSelectedSpecies() {
        // (1) get selected Species
        PluginListOf listOf = plug.getSelectedSpeciesNode();

        if(listOf != null){
                // get PluginSpeciesAlias
                PluginSpeciesAlias alias = (PluginSpeciesAlias)listOf.get(0);

                //(2) get position
                double pos_x = alias.getX();
                double pos_y = alias.getY();

                //(3) get Species
                PluginSpecies sp = alias.getSpecies();

                //Show species information
                textName.setText(sp.getName());
                textId.setText(sp.getId());
                textX.setText(String.valueOf(pos_x).toString());
                textY.setText(String.valueOf(pos_y).toString());
        }
}
```

(1) You can use the “getSelectedSpeciesNode” method in Plugin class to get properties of the species selected on CellDesigner. A return value isn’t a list of species but a list of speciesAlias. In CellDesigner, species which have different drawing information such as colors and positions can be displayed on canvas even if they have the same speciesID. In this case, SpeciesAlias is used to identify the species selected on canvas.

(2) Get the position from SpeciesAlias class.

(3) Get Species class from SpeciesAlias to get the name of species.

8.2 Updating PluginSpecies

If you need to update the class like the PluginSpecies (which extends PluginSBase) based on an event that are passed by CellDesigner, use the "update" method in the class instead of updating the properties one by one. The following code is an example of updating a species.

```
public void SBaseChanged(PluginSBase sbase) {
        if(sbase instanceof PluginSpecies){
              PluginSpecies sp = PluginSampleDialog.model.getSpecies(((PluginSpecies)sbase).getId());
              if(sp != null){
                  sp.update((PluginSpecies) sbase);
              }
        }
}
```

8.3 Creating a new species and notifying CellDesigner of the event

(1) Get an SBML model as a PluginModel class from CellDesigner
(2) Instantiate PluginSpecies class to create a new Species. "Generic Protein" is created in this source.
(3) Set positions to SpeciesAlias for display location
(4) Set the species to the PluginModel, and notify CellDesigner of this species as PluginSBase. If plugin doesn't notify, CellDesigner can't reflect these changes.

```
/**
 *   Add new Species
 *
 */
private void addSpecies() {
        //(1) get selected model
        PluginModel model = plug.getSelectedModel();

        //get name of Species
        String name = textName.getText();

        // (2) create PluginSBase
        PluginSBase sbase
            = new PluginSpecies(PluginSpeciesSymbolType.PROTEIN_GENERIC, name);


        // (3) set position
        String x = textX.getText();
        String y = textY.getText();

        try{
                double pos_x = Double.valueOf(x).doubleValue();
                double pos_y = Double.valueOf(y).doubleValue();


                PluginSpecies species = (PluginSpecies)sbase;

                PluginSpeciesAlias psa = species.getSpeciesAlias(0);
                psa.setFramePosition(pos_x,pos_y);

                // (4) set species into model
                model.addSpecies(species);
                plug.notifySBaseAdded((PluginSBase)species);

                //Add residue
                PluginProtein protein = psa.getProtein();
                PluginModificationResidue residue =
                                new PluginModificationResidue(protein);
                residue.setName("P");
                protein.addPluginModificationResidue(residue);
                plug.notifySBaseChanged(residue);

                //add modification
                PluginModification m = new PluginModification();
                m.setResidue(residue.getId());
                m.setState(PluginModification.STATE_PHOSPHORYLATED);
                psa.getModifications().append(m);

                plug.notifySBaseChanged(psa);

                textX.setText(null);
                textY.setText(null);

        }catch(NumberFormatException e){
                JOptionPane.showMessageDialog(
                                this, "Input number!!","error",
                                JOptionPane.ERROR_MESSAGE
                                );

        }
}
```

8.4 Simulate a model from Plugin

(1) Get an SBML model as a PluginModel class from CellDesigner
(2) Instantiate PluginSimulation class with obtained PluginModel.
(3) Set simulation condition. For example, simulation time (endTime), num. of steps and output file can be specified by setter methods.
(4) Execute a simulation. Information Dialog will be shown when the simulation is completed.

```
/**
  *    Run Simulation
  *
  */
private void runSimulation(String filename) {
        // (1) Get selected model
        PluginModel model = plugin.getSelectedModel();

        // (2) Instantiate PluginSimulation class with obtained PluginModel
        PluginSimulation psim = new PluginSimulation(model);

        // (3) Set simulation condition (simulation time, num. of steps, output file).
        psim.setEndTime(4000);
        psim.setNumOfSteps(1000);
        psim.setOutputFile(filename);

        // (4) Execute a simulation.
        psim.runSimulation();
}
```

8.5 Export graphical notation to SBGN-ML

(1) Get an SBML model as a PluginModel class from CellDesigner
(2) Instantiate File class which will be used to store SBGN-ML.
(3) Instantiate PluginSBGNML class with obtained PluginModel and File.
(4) Export graphical notation to SBGN-ML. Information Dialog will be shown when the export procedure is completed.

If there is no output file specified (if you skip procedure (2)), then CellDesigner will launch File Dialog to ask users where to store the exported SBGN-ML.

```
/**
 *   Export to SBGN-ML
 *
 */
private void runSimulation(String filename) {
        // (1) Get selected model
        PluginModel model = plugin.getSelectedModel();

        // (2) Instantiate File class
        File sbgnFile = new File(“foo.sbgn”);

        // (3) Instantiate PluginSimulation class with obtained PluginModel
        PluginSBGNML psbgn = new PluginSBGNML(model, sbgnFile);

        // (4) Export graphical notation to SBGN-ML
        psbgn.export();
}
```