JustClust Plug-in Development Manual

Contents

1. Writing a Parsing Plug-in

To write a parsing plug-in for JustClust, you need to have a class which implements the FileParserPluginInterface interface from JustClust (justclust.plugins.parsing.FileParserPluginInterface).
The class which implements the FileParserPluginInterface interface needs to be contained in a jar file.
The class should have the same directory structure within the jar file as its package structure (e.g. the class package.subpackage.class should have the file path package/subpackage/class within the jar file).
The jar file is your plug-in and can contain other classes which are used by the class which implements the FileParserPluginInterface interface.

The FileParserPluginInterface interface requires the implementing class to have six methods.
These are explained below.

1.1. The getFileType Method

The getFileType method should return a String.
This String is displayed to the user as the file type(s) which the plug-in can parse.

1.2. The getDescription Method

The getDescription method should return a String.
This String is displayed to the user as a description of the plug-in.

1.3. The getConfigurationControls Method

The getConfigurationControls method should return an ArrayList of plug-in configuration controls, which are instances of classes which implement PluginConfigurationControlInterface (justclust.plugins.configurationcontrols.PluginConfigurationControlInterface).
These controls are shown to the user to allow them to input information which your plug-in can read and act on accordingly, effectively allowing the user to configure your plug-in.
If you do not want your plug-in to be configurable by the user, simply return an empty ArrayList of type PluginConfigurationControlInterface.

There are currently four types of plug-in configuration controls: CheckBoxControls (justclust.plugins.configurationcontrols.CheckBoxControl), ComboBoxControls (justclust.plugins.configurationcontrols.ComboBoxControl), FileSystemPathControls (justclust.plugins.configurationcontrols.FileSystemPathControl), and TextFieldControls (justclust.plugins.configurationcontrols.TextFieldControl).
all of four classes implement the PluginConfigurationControlInterface interface and so can be added to the ArrayList of type PluginConfigurationControlInterface which is to be returned.

Before you return the ArrayList, you may want to specify a few things about the plug-in configuration controls within by assigning values to their fields.

CheckBoxControls have two fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The checked field contains a boolean which is true if the CheckBoxControl should be checked initially (as a default value).

ComboBoxControls have three fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The options field contains an ArrayList of Strings each of which is one of the options which the user can select from in the ComboBoxControl.
- The selectedOptionIndex field contains an int which is the index of the String in the options field ArrayList which is initially selected (as a default value).

FileSystemPathControls have three fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The text field contains a String which is the text which is initially within the control (as a default value).
- The directoriesOnly field contains a boolean which is true if the file system path specified by the user should be a path to a directory rather than a file.

TextFieldControls have two fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The text field contains a String which is the text which is initially within the control (as a default value).

Once the user has input into the plug-in configuration controls which you specified in this method, and has clicked on the 'Create Network' button, the user's input will be made accessible to your plug-in via the fields of the plug-in configuration controls.
The checked field of CheckBoxControls is updated to true if the user checked the CheckBoxControl, and false if the user unchecked the CheckBoxControl.
The selectedOptionIndex field of ComboBoxControls is updated to the index of the String in the options field ArrayList which is selected by the user.
The text field of FileSystemPathControls is updated to the text which is input into the control by the user.
The text field of TextFieldControls is updated to the text which is input into the control by the user.

1.4. The parseFile Method

The parseFile method takes as parameters a File (java.io.File), which is the file to be parsed; an ArrayList of Nodes (justclust.datastructures.Node); and an ArrayList of Edges (justclust.datastructures.Edge).
The method should update the ArrayList of Nodes with Nodes which each represent a node in the network being created.
The method should also update the ArrayList of Edges with Edges which each represent an edge in the network being created.

Nodes have two fields which should be initialised by parsing plug-ins (other fields can be ignored).
- The label field contains a String which is the label of the node.
- The microarrayValues field contains an ArrayList of Doubles which are the microarray values for the Node.

These values should appear in the ArrayList in the order in which they are parsed. If your plug-in does not parse microarray data this field can be ignored.

Edges have three fields which should be initialised by parsing plug-ins (other fields can be ignored).
- The node1 field contains a Node which is one of the two nodes which the edge connects.
- The node2 field contains a Node which is the other of the two nodes which the edge connects.
- The weight field contains a double which is the weight of the edge.

1.5. The isMicroarrayData Method

The isMicroarrayData method should return a boolean.
If this boolean is true, JustClust will allow the user to see a microarray heat map of the data which is parsed by your plug-in.
If this boolean is false, the microarray heat map will not be available to the user.

1.6. The getMicroarrayHeaders Method

The getMicroarrayHeaders method should return an ArrayList of Strings.
These Strings should be the headers of the columns in the microarray data.
The values should appear in the ArrayList in the order in which they are parsed.
If your plug-in does not parse microarray data, this method should return null.

1.7. Example Code

The following code is from the delimiter-separated values file parser plug-in.
the delimiter-separated values file parser plug-in parses files containing values seperated by any delimiter, such as comma-separated values (.csv) files and tab-separated values (.tsv, .tab) files.

```
public class DelimiterSeparatedValuesFileParser implements FileParserPluginInterface {

    // comboBoxControl allows the getConfigurationControls and parseFile
    // methods to share the selectedOptionIndex field of this comboBoxControl
    public ComboBoxControl comboBoxControl;

    public String getFileType() throws Exception {
        return "Delimiter-separated values file (.csv, .tab, .tsv)";
    }

    public String getDescription() throws Exception {
        return "This file parser plug-in parses files containing values seperated by any delimiter, such as comma-
separated values (.csv) files and tab-separated values (.tsv, .tab) files.";
    }

    public ArrayList<PluginConfigurationControlInterface> getConfigurationControls() throws Exception {

        comboBoxControl = new ComboBoxControl();
        comboBoxControl.label = "Delimiter Between Values:";
        comboBoxControl.options = new ArrayList<String>();
        comboBoxControl.options.add("Commas");
        comboBoxControl.options.add("Tabs");
        comboBoxControl.selectedOptionIndex = 0;

        ArrayList<PluginConfigurationControlInterface> controls = new
ArrayList<PluginConfigurationControlInterface>();
        controls.add(comboBoxControl);

        return controls;

    }

    public void parseFile(File file, ArrayList<Node> networkNodes, ArrayList<Edge> networkEdges) throws
Exception {

        // This code creates the networkEdges and networkNodes data
        // structures with the contents of a file.
        Scanner scanner = new Scanner(file);
        Hashtable<String, Node> hashTable = new Hashtable<String, Node>();
        while (scanner.hasNextLine()) {

            Scanner lineScanner = new Scanner(scanner.nextLine().trim());
            // the delimiter chosen by the user was commas
            if (comboBoxControl.options.get(comboBoxControl.selectedOptionIndex).equals("Commas")) {
                lineScanner.useDelimiter(",");
            }
            // the delimiter chosen by the user was tabs
            if (comboBoxControl.options.get(comboBoxControl.selectedOptionIndex).equals("Tabs")) {
                lineScanner.useDelimiter("\t");
            }
            Edge edge = new Edge();
            networkEdges.add(edge);

            String identifier = lineScanner.next().trim();
            if (hashTable.containsKey(identifier)) {
                edge.node1 = hashTable.get(identifier);
            } else {
                edge.node1 = new Node();
                edge.node1.label = identifier;
                hashTable.put(identifier, edge.node1);
            }

            identifier = lineScanner.next().trim();
            if (hashTable.containsKey(identifier)) {
                edge.node2 = hashTable.get(identifier);
            } else {
                edge.node2 = new Node();
                edge.node2.label = identifier;
                hashTable.put(identifier, edge.node2);
            }

            edge.weight = Double.valueOf(lineScanner.next().trim());

            lineScanner.close();

        }
        ArrayList<Node> arrayList = new ArrayList<Node>(hashTable.values());
        for (Node node : arrayList) {
            networkNodes.add(node);
        }
        scanner.close();

        // This code sorts the Edges in the networkEdges data
        // structure from largest weight field to smallest weight field.
        for (int i = 1; i < networkEdges.size(); i++) {
            Edge edge = networkEdges.get(i);
            int j;
            for (j = i - 1; j >= 0
                    && edge.weight > networkEdges.get(j).weight; j--) {
                networkEdges.set(j + 1, networkEdges.get(j));
            }
            networkEdges.set(j + 1, edge);
        }

        // This code creates, for each Edge, the Node.edges data structure of
        // each of its two Nodes.
        for (Node node : networkNodes) {
            node.edges = new ArrayList<Edge>();
        }
        for (Edge edge : networkEdges) {
            if (!edge.node1.edges.contains(edge)) {
```

```
                edge.node1.edges.add(edge);
            }
            if (!edge.node2.edges.contains(edge)) {
                edge.node2.edges.add(edge);
            }
        }

    }

    public boolean isMicroarrayData() throws Exception {
        return false;
    }

    public ArrayList<String> getMicroarrayHeaders() throws Exception {
        return null;
    }
}
```

2. Writing a Clustering Plug-in

To write a clustering plug-in for JustClust, you need to have a class which implements the ClusteringAlgorithmPluginInterface interface from JustClust (justclust.plugins.clustering.ClusteringAlgorithmPluginInterface).
The class which implements the ClusteringAlgorithmPluginInterface interface needs to be contained in a jar file.
The class should have the same directory structure within the jar file as its package structure (e.g. the class package.subpackage.class should have the file path package/subpackage/class within the jar file).
The jar file is your plug-in and can contain other classes which are used by the class which implements the ClusteringAlgorithmPluginInterface interface.

The ClusteringAlgorithmPluginInterface interface requires the implementing class to have six methods.
These are explained below.

2.1. The getName Method

The getName method should return a String.
This String is displayed to the user as the name of the plug-in.

2.2. The getDescription Method

The getDescription method should return a String.
This String is displayed to the user as a description of the plug-in.

2.3. The getConfigurationControls Method

The getConfigurationControls method should return an ArrayList of plug-in configuration controls, which are instances of classes which implement PluginConfigurationControlInterface (justclust.plugins.configurationcontrols.PluginConfigurationControlInterface).
These controls are shown to the user to allow them to input information which your plug-in can read and act on accordingly, effectively allowing the user to configure your plug-in.
If you do not want your plug-in to be configurable by the user, simply return an empty ArrayList of type

PluginConfigurationControlInterface.

There are currently four types of plug-in configuration controls: CheckBoxControls (justclust.plugins.configurationcontrols.CheckBoxControl), ComboBoxControls (justclust.plugins.configurationcontrols.ComboBoxControl), FileSystemPathControls (justclust.plugins.configurationcontrols.FileSystemPathControl), and TextFieldControls (justclust.plugins.configurationcontrols.TextFieldControl).
all of four classes implement the PluginConfigurationControlInterface interface and so can be added to the ArrayList of type PluginConfigurationControlInterface which is to be returned.

Before you return the ArrayList, you may want to specify a few things about the plug-in configuration controls within by assigning values to their fields.

CheckBoxControls have two fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The checked field contains a boolean which is true if the CheckBoxControl should be checked initially (as a default value).

ComboBoxControls have three fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The options field contains an ArrayList of Strings each of which is one of the options which the user can select from in the ComboBoxControl.
- The selectedOptionIndex field contains an int which is the index of the String in the options field ArrayList which is initially selected (as a default value).

FileSystemPathControls have three fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The text field contains a String which is the text which is initially within the control (as a default value).
- The directoriesOnly field contains a boolean which is true if the file system path specified by the user should be a path to a directory rather than a file.

TextFieldControls have two fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The text field contains a String which is the text which is initially within the control (as a default value).

Once the user has input into the plug-in configuration controls which you specified in this method, and has clicked on the 'Create Network' button, the user's input will be made accessible to your plug-in via the fields of the plug-in configuration controls.
The checked field of CheckBoxControls is updated to true if the user checked the CheckBoxControl, and false if the user unchecked the CheckBoxControl.
The selectedOptionIndex field of ComboBoxControls is updated to the index of the String in the options field ArrayList which is selected by the user.
The text field of FileSystemPathControls is updated to the text which is input into the control by the user.
The text field of TextFieldControls is updated to the text which is input into the control by the user.

2.4. The clusterNetwork Method

The clusterNetwork method takes as parameters an ArrayList of Nodes (justclust.datastructures.Node), which are the nodes in the current network; an ArrayList of Edges (justclust.datastructures.Edge), which are the edges in the current current; and an ArrayList of Clusters (justclust.datastructures.Cluster).
The method should update the ArrayList of Clusters with Clusters which each represent a cluster in the network being clustered.

Nodes have one field which should be accessed but not changed by clustering plug-ins (other fields can be ignored).

- The edges field contains an ArrayList of Edges each of which is an edge which connects the node to another.

Edges have three fields which should be accessed but not changed by clustering plug-ins (other fields can be ignored).

- The node1 field contains a Node which is one of the two nodes which the edge connects.
- The node2 field contains a Node which is the other of the two nodes which the edge connects.
- The weight field contains a double which is the weight of the edge.

Clusters have one field which should be initialised by clustering plug-ins (other fields can be ignored).

- The nodes field contains an ArrayList of Nodes which are contained by the cluster.

2.5. The isHierarchicalClustering Method

The isHierarchicalClustering method should return a boolean.
If this boolean is true, JustClust will allow the user to see a dendrogram of the clustering which is created by your plug-in.
If this boolean is false, the dendrogram will not be available to the user.

2.6. The getRootDendrogramClusters Method

The getRootDendrogramClusters method should return an ArrayList of DendrogramClusters (justclust.toolbar.dendrogram.DendrogramCluster).
These DendrogramClusters should each be the root of a hierarchical tree in the current clustering.
If the clustered network was originally fully connected, there will be only one hierarchical tree in the current clustering.
If your plug-in does not create hierarchical clusterings, this method should return null.

2.7. Example Code

The following code is from the single-linkage clustering algorithm plug-in.
the single-linkage clustering algorithm plug-in clusters the current network with a single-linkage clustering algorithm.

```
/**
 * This class contains a method which performs a single linkage clustering
 * algorithm.
 */
public class SingleLinkageClusteringAlgorithm implements
        ClusteringAlgorithmPluginInterface {

    // clusterAmountTextFieldControl and minimumEdgeWeightTextFieldControl allow
    // the getConfigurationControls and clusterNetwork methods to share the text
    // fields of these TextFieldControls
    public TextFieldControl clusterAmountTextFieldControl;
    public TextFieldControl minimumEdgeWeightTextFieldControl;
    public ArrayList<DendrogramCluster> rootDendrogramClusters;

    /**
     * This method returns a display name for the clustering algorithm which a
     * method of this class performs.
     */
    public String getName() throws Exception {
        return "Single-linkage clustering algorithm";
    }

    public String getDescription() throws Exception {
        return "This clustering algorithm plug-in clusters the current network with a single-linkage clustering
algorithm.";
    }

    public ArrayList<PluginConfigurationControlInterface> getConfigurationControls() throws Exception {

        ArrayList<PluginConfigurationControlInterface> controls = new
ArrayList<PluginConfigurationControlInterface>();

        clusterAmountTextFieldControl = new TextFieldControl();
        clusterAmountTextFieldControl.label = "Number of Clusters:";
        clusterAmountTextFieldControl.text = "50";
        controls.add(clusterAmountTextFieldControl);

        minimumEdgeWeightTextFieldControl = new TextFieldControl();
        minimumEdgeWeightTextFieldControl.label = "Minimum Edge Weight for Combining Clusters:";
        minimumEdgeWeightTextFieldControl.text = "0";
        controls.add(minimumEdgeWeightTextFieldControl);

        return controls;

    }

    /**
     * This method performs a single linkage clustering algorithm.
     */
    public void clusterNetwork(ArrayList<Node> networkNodes, ArrayList<Edge> networkEdges,
ArrayList<Cluster> networkClusters) throws Exception {

        ArrayList<DendrogramCluster> dendrogramClusters = new ArrayList<DendrogramCluster>();

        // This code creates a Cluster for each of the Nodes in the
        // networkNodes data structure. These clusters will be merged
        // during the single linkage clustering algorithm.
        for (Node node : networkNodes) {

            Cluster cluster = new Cluster();
            networkClusters.add(cluster);
    cluster.nodes = new ArrayList<Node>();
    cluster.nodes.add(node);
    node.cluster = cluster;

    DendrogramCluster dendrogramCluster = new DendrogramCluster();
    dendrogramCluster.distance = 0;
    dendrogramCluster.left = null;
    dendrogramCluster.right = null;
    dendrogramCluster.nodeIndex = networkNodes.indexOf(node);
    dendrogramClusters.add(dendrogramCluster);

}

// This code creates a data structure of Edges called edges which is a
// shallow copy of the networkEdges data structure. The Edges in
// the edges data structure are sorted from largest weight field to
// smallest weight field.
ArrayList<Edge> edges = (ArrayList<Edge>) networkEdges.clone();
for (int i = 1; i < edges.size(); i++) {
    Edge edge = edges.get(i);
    int j;
    for (j = i - 1; j >= 0
            && edge.weight > edges.get(j).weight; j--) {
        edges.set(j + 1, edges.get(j));
    }
    edges.set(j + 1, edge);
}

// This code removes Edges from the edges data structure which link a
// Node to itself.
for (int i = 0; i < edges.size(); i++) {
    if (edges.get(i).node1 == edges.get(i).node2) {
        edges.remove(i);
        i--;
    }
}

// This code removes Edges from the edges data structure which link
// Nodes which are already linked by another Edge in the edges data
// structure.
for (int i = 0; i < edges.size(); i++) {
    for (int j = i + 1; j < edges.size(); j++) {
        if (edges.get(i).node1 == edges.get(j).node1
                && edges.get(i).node2 == edges.get(j).node2
                || edges.get(i).node1 == edges.get(j).node2
                && edges.get(i).node2 == edges.get(j).node1) {
            edges.remove(j);
            j--;
        }
    }
}
```

```
// This code clusters the Nodes in the networkNodes data structure.
// At each stage in the single linkage clustering algorithm, the closest
// two Clusters are merged.
int i = 0;
while (networkClusters.size() > Integer.parseInt(clusterAmountTextFieldControl.text)
        && edges.get(i).weight >= Integer.parseInt(minimumEdgeWeightTextFieldControl.text)) {
    if (edges.get(i).node1.cluster != edges.get(i).node2.cluster) {

        DendrogramCluster dendrogramCluster = new DendrogramCluster();
        dendrogramCluster.distance = edges.get(i).weight;
        dendrogramCluster.left =
                dendrogramClusters.get(networkClusters.indexOf(edges.get(i).node1.cluster));;
        dendrogramCluster.right =
                dendrogramClusters.get(networkClusters.indexOf(edges.get(i).node2.cluster));;
        dendrogramClusters.remove(networkClusters.indexOf(edges.get(i).node1.cluster));
        dendrogramClusters.add(networkClusters.indexOf(edges.get(i).node1.cluster), dendrogramCluster);
        dendrogramClusters.remove(networkClusters.indexOf(edges.get(i).node2.cluster));

        edges.get(i).node1.cluster.nodes.addAll(edges.get(i).node2.cluster.nodes);
        networkClusters.remove(edges.get(i).node2.cluster);
        for (Node node : edges.get(i).node2.cluster.nodes) {
            node.cluster = edges.get(i).node1.cluster;
        }

    }
    i++;
}

ArrayList<Node> networkNodesCopy = new ArrayList<Node>();
for (Node node : networkNodes) {
    Node nodeCopy = new Node();
    networkNodesCopy.add(nodeCopy);
}
ArrayList<Edge> networkEdgesCopy = new ArrayList<Edge>();
for (Edge edge : edges) {
    Edge edgeCopy = new Edge();
    edgeCopy.node1 = networkNodesCopy.get(networkNodes.indexOf(edge.node1));
    edgeCopy.node2 = networkNodesCopy.get(networkNodes.indexOf(edge.node2));
    edgeCopy.weight = edge.weight;
    networkEdgesCopy.add(edgeCopy);
}
ArrayList<Cluster> networkClustersCopy = new ArrayList<Cluster>();
for (Cluster cluster : networkClusters) {
    Cluster clusterCopy = new Cluster();
    clusterCopy.nodes = new ArrayList<Node>();
    for (Node node : cluster.nodes) {
        Node nodeCopy = networkNodesCopy.get(networkNodes.indexOf(node));
        nodeCopy.cluster = clusterCopy;
        clusterCopy.nodes.add(nodeCopy);
    }
```

```
            networkClustersCopy.add(clusterCopy);
        }

        while (networkClustersCopy.size() > 1 && i < networkEdgesCopy.size()) {
            if (networkEdgesCopy.get(i).node1.cluster != networkEdgesCopy.get(i).node2.cluster) {

                DendrogramCluster dendrogramCluster = new DendrogramCluster();
                dendrogramCluster.distance = networkEdgesCopy.get(i).weight;
                dendrogramCluster.left =
                    dendrogramClusters.get(networkClustersCopy.indexOf(networkEdgesCopy.get(i).node1.cluster));
                dendrogramCluster.right =
                    dendrogramClusters.get(networkClustersCopy.indexOf(networkEdgesCopy.get(i).node2.cluster));
                dendrogramClusters.remove(networkClustersCopy.indexOf(networkEdgesCopy.get(i).node1.cluster));
                dendrogramClusters.add(networkClustersCopy.indexOf(networkEdgesCopy.get(i).node1.cluster),
  dendrogramCluster);
                dendrogramClusters.remove(networkClustersCopy.indexOf(networkEdgesCopy.get(i).node2.cluster));

                networkEdgesCopy.get(i).node1.cluster.nodes.addAll(networkEdgesCopy.get(i).node2.cluster.nodes);
                networkClustersCopy.remove(networkEdgesCopy.get(i).node2.cluster);
                for (Node node : networkEdgesCopy.get(i).node2.cluster.nodes) {
                    node.cluster = networkEdgesCopy.get(i).node1.cluster;
                }

            }
            i++;
        }

        rootDendrogramClusters = dendrogramClusters;

    }

    public boolean isHierarchicalClustering() {
        return true;
    }

    public ArrayList<DendrogramCluster> getRootDendrogramClusters() {
        return rootDendrogramClusters;
    }
}
```

3. Writing a Visualisation Plug-in

To write a visualisation plug-in for JustClust, you need to have a class which implements the VisualisationLayoutPluginInterface interface from JustClust (justclust.plugins.visualisation.VisualisationLayoutPluginInterface).
The class which implements the VisualisationLayoutPluginInterface interface needs to be contained in a jar file. The class should have the same directory structure within the jar file as its package structure (e.g. the class package.subpackage.class should have the file path package/subpackage/class within the jar file).
The jar file is your plug-in and can contain other classes which are used by the class which implements the VisualisationLayoutPluginInterface interface.

The VisualisationLayoutPluginInterface interface requires the implementing class to have four methods.
These are explained below.

3.1. The getName Method

The getName method should return a String.
This String is displayed to the user as the name of the plug-in.

3.2. The getDescription Method

The getDescription method should return a String.
This String is displayed to the user as a description of the plug-in.

3.3. The getConfigurationControls Method

The getConfigurationControls method should return an ArrayList of plug-in configuration controls, which are instances of classes which implement PluginConfigurationControlInterface (justclust.plugins.configurationcontrols.PluginConfigurationControlInterface).
These controls are shown to the user to allow them to input information which your plug-in can read and act on accordingly, effectively allowing the user to configure your plug-in.
If you do not want your plug-in to be configurable by the user, simply return an empty ArrayList of type PluginConfigurationControlInterface.

There are currently four types of plug-in configuration controls: CheckBoxControls (justclust.plugins.configurationcontrols.CheckBoxControl), ComboBoxControls (justclust.plugins.configurationcontrols.ComboBoxControl), FileSystemPathControls (justclust.plugins.configurationcontrols.FileSystemPathControl), and TextFieldControls (justclust.plugins.configurationcontrols.TextFieldControl).
all of four classes implement the PluginConfigurationControlInterface interface and so can be added to the ArrayList of type PluginConfigurationControlInterface which is to be returned.

Before you return the ArrayList, you may want to specify a few things about the plug-in configuration controls within by assigning values to their fields.

CheckBoxControls have two fields.
- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The checked field contains a boolean which is true if the CheckBoxControl should be checked initially (as a default value).

ComboBoxControls have three fields.
- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The options field contains an ArrayList of Strings each of which is one of the options which the user can select from in the ComboBoxControl.
- The selectedOptionIndex field contains an int which is the index of the String in the options field ArrayList which is initially selected (as a default value).

FileSystemPathControls have three fields.
- The label field contains a String which is the label which will appear before the control to describe what the

user should input into the control.

- The text field contains a String which is the text which is initially within the control (as a default value).
- The directoriesOnly field contains a boolean which is true if the file system path specified by the user should be a path to a directory rather than a file.

TextFieldControls have two fields.

- The label field contains a String which is the label which will appear before the control to describe what the user should input into the control.
- The text field contains a String which is the text which is initially within the control (as a default value).

Once the user has input into the plug-in configuration controls which you specified in this method, and has clicked on the 'Create Network' button, the user's input will be made accessible to your plug-in via the fields of the plug-in configuration controls.
The checked field of CheckBoxControls is updated to true if the user checked the CheckBoxControl, and false if the user unchecked the CheckBoxControl.
The selectedOptionIndex field of ComboBoxControls is updated to the index of the String in the options field ArrayList which is selected by the user.
The text field of FileSystemPathControls is updated to the text which is input into the control by the user.
The text field of TextFieldControls is updated to the text which is input into the control by the user.

3.4. The applyLayout Method

The applyLayout method should use the getXCoordinate, setXCoordinate, getYCoordinate, and setYCoordinate methods of Nodes (justclust.datastructures.Node) in the networkNodes field of the Data class from JustClust (justclust.Data) to reposition the Nodes in the graphical representation of the network.
The method can access the networkNodes, networkEdges, and networkClusters fields of the Data class to determine where the Nodes should be placed.
These contain ArrayLists of Nodes, Edges (justclust.datastructures.Edge), and Clusters (justclust.datastructures.Cluster) respectively.

Nodes have one field which should be accessed but not changed by visualisation plug-ins (other fields can be ignored).

- The edges field contains an ArrayList of Edges each of which is an edge which connects the node to another.
- The cluster field contains the Cluster which the node belongs to.

Edges have three fields which should be accessed but not changed by visualisation plug-ins (other fields can be ignored).

- The node1 field contains a Node which is one of the two nodes which the edge connects.
- The node2 field contains a Node which is the other of the two nodes which the edge connects.
- The weight field contains a double which is the weight of the edge.

Clusters have one field which should be accessed but not changed by visualisation plug-ins (other fields can be ignored).

- The nodes field contains an ArrayList of Nodes which are contained by the cluster.

3.5. Example Code

The following code is from the Cytoscape grid visualisation layout plug-in.
the Cytoscape grid visualisation layout plug-in lays-out the graphical representation of the current network with the grid layout from Cytoscape.

```
public class CytoscapeGridVisualisationLayout implements VisualisationLayoutPluginInterface {

    // textFieldControl allows the getConfigurationControls and applyLayout
    // methods to share the text field of this TextFieldControl
    public TextFieldControl horizontalSpacingControl;
    public TextFieldControl verticalSpacingControl;

    public String getName() throws Exception {
        return "Cytoscape grid visualisation layout";
    }

    public String getDescription() throws Exception {
        return "This visualisation layout plug-in lays-out the graphical representation of the current network with the
grid layout from Cytoscape.";
    }

    public ArrayList<PluginConfigurationControlInterface> getConfigurationControls() throws Exception {

        horizontalSpacingControl = new TextFieldControl();
        horizontalSpacingControl.label = "Horizontal Node Spacing:";
        horizontalSpacingControl.text = "50";

        verticalSpacingControl = new TextFieldControl();
        verticalSpacingControl.label = "Vertical Node Spacing:";
        verticalSpacingControl.text = "40";

        ArrayList<PluginConfigurationControlInterface> controls = new
ArrayList<PluginConfigurationControlInterface>();
        controls.add(horizontalSpacingControl);
        controls.add(verticalSpacingControl);

        return controls;

    }

    public void applyLayout(ArrayList<Node> networkNodes, ArrayList<Edge> networkEdges,
ArrayList<Cluster> networkClusters) throws Exception {

        double nodeHorizontalSpacing = Integer.parseInt(horizontalSpacingControl.text);
        double nodeVerticalSpacing = Integer.parseInt(verticalSpacingControl.text);

        double currX = 0.0d;
        double currY = 0.0d;
        double initialX = 0.0d;
        double initialY = 0.0d;

        // Yes, our size and starting points need to be different

        // the nodeCount is set to 0 so that, if there are no nodes, the rest
        // of this method will not try to access nodes which don't exist,
        // causing an error.
        // networkNodes != null is checked so that
        // networkNodes.size() is not called when
        // networkNodes == null which will cause an
        // error.
        int nodeCount = 0;
        if (networkNodes != null) {
            nodeCount = networkNodes.size();
        }
        final int columns = (int) Math.sqrt(nodeCount);

        // Calculate our starting point as the geographical center of the
        // selected nodes.
        for (int i = 1; i < networkNodes.size(); i++) {
            initialX += (networkNodes.get(i).getGraphicalNodeXCoordinate() / nodeCount);
            initialY += (networkNodes.get(i).getGraphicalNodeYCoordinate() / nodeCount);
        }

        // initialX and initialY reflect the center of our grid, so we
        // need to offset by distance*columns/2 in each direction
        initialX = initialX - ((nodeHorizontalSpacing * (columns - 1)) / 2);
        initialY = initialY - ((nodeVerticalSpacing * (columns - 1)) / 2);
        currX = initialX;
        currY = initialY;

        int count = 0;

        // Set visual property.
        // TODO: We need batch apply method for Visual Property values for
        // performance.
        for (int i = 0; i < networkNodes.size(); i++) {
            Node node = networkNodes.get(i);
            node.setGraphicalNodeXCoordinate(currX);
            node.setGraphicalNodeYCoordinate(currY);

            count++;

            if (count == columns) {
                count = 0;
                currX = initialX;
                currY += nodeVerticalSpacing;
            } else {
                currX += nodeHorizontalSpacing;
            }
        }

    }
}
```