TASSEL 5.0 Self-Describing Plugins:

Guide to creating Tassel Plugins

Terry Casstevens (tmc46@cornell.edu), Jason Wallace

October 7, 2016

Prerequisites / Source Code

Java SDK 8.0 or later (http://java.sun.com/javase/downloads/index.jsp).

`git clone https://bitbucket.org/tasseladmin/tassel-5-source.git`

All Plugin classes *should*:

- Extend net.maizegenetics.plugindef.AbstractPlugin
- Implement this constructor:

```
public PluginName(Frame parentFrame, boolean isInteractive) {
       super(parentFrame, isInteractive);
}
```

- Define PluginParameters to hold the parameter values (*See details below*)
- Implement the Plugin.processData() method for performing the plugin's function. :

```
public DataSet processData(DataSet input) {
      //do whatever your plugin does
      return result;  // Note: this can return null
}
```

- Override the three abstract classes for use in the GUI. (Please fill in something meaningful for getButtonName() and getToolTipText()):

```
@Override
public ImageIcon getIcon() {
    return null;
}

@Override
public String getButtonName() {
    return "Example Function";
}

@Override
public String getToolTipText() {
    return "Performs Function by Algorithm";
}
```

All Plugin classes *may:*

- Optionally implement AbstractPlugin.preProcessParameters() (See details below)
- Optionally implement AbstractPlugin.postProcessParameters() (See details below)
- Optionally implement Plugin.pluginDescription() method

```
@Override
public String pluginDescription() {
    return "This plugin takes a TASSEL-generated HDF5 file and outputs various " +
    "summary statistics such as the number of taxa, their names, etc.";
```

```
}
```

- Optionally implement Plugin.getCitation() method

```
@Override
public String getCitation() {
    return "Bradbury PJ, Zhang Z, Kroon DE, Casstevens TM, Ramdoss Y, Buckler ES.
    (2007) TASSEL: Software for association mapping of complex traits in diverse
    samples. Bioinformatics 23:2633-2635.";
}
```

- Optionally implement Plugin.pluginUserManualURL() method

```
@Override
public String pluginUserManualURL() {
    return "https://bitbucket.org/tasseladmin/tassel-5-source/wiki/UserManual/Kinship/Kinship";
}
```

All Plugin classes *should not:*

- Call System.exit() at any point
- Implement the Plugin.performFunction() method
- Handle error situations (i.e. showing dialog, output logging, etc.). Throw an exception, and AbstractPlugin will handle it.

```
// Don't do this...
if (alignInList.size() != 1) {
    String gpMessage = "Invalid selection.  Please select one genotype alignment.";
    if (isInteractive()) {
        JOptionPane.showMessageDialog(getParentFrame(), gpMessage);
    } else {
        myLogger.error(gpMessage);
    }
    return null;
}

// Do this...
if (alignInList.size() != 1) {
    throw new IllegalArgumentException("Invalid selection.  Please select one
genotype alignment.");
}
```

- Implement a main() class (“static public void main()”)
    - *Note: A main() class is required to automatically generate PluginParameter getters and setters (below), but it should be commented out before uploading the class to the repository.*

Defining PluginParameters

Step 1 – Declare each parameter

Declare each parameter as a private variable within your plugin using the PluginParameter.Builder class. (The order you define them in will determine the order they show up in the GUI dialog box, so try to group them logically.) Example:

```
private PluginParameter<String> myVarName = new
PluginParameter.Builder<>("varName", null, String.class).build();
```

The data type (String, Double, Enum, etc.) is defined with the diamond operator (<>), and the constructor takes three arguments:

- The name of the parameter for the command line (no spaces; use camelCase to connect multiple words)
- The default value for the parameter ("null" if no default)
- The class type of the parameter value (eg., String.class, Double.class, etc.)

After the constructor, call the various Builder commands to set further values of the PluginParameter (see below), with the build() command coming last of all. Although these can be done as separate commands, it's better coding style to string them together. For readability, it's often a good idea to put each on a separate line. Example:

```
private PluginParameter<String> inputFile =
    new PluginParameter.Builder<>("inputFile", null, String.class)
    .inFile().required(true).build();
```

This is equivalent to (but cleaner than) the following code:

```
private PluginParameter.Builder<String> inputFileBuilder =
    new PluginParameter.Builder<>("inputFile", null, String.class);
inputFileBuilder.inFile();
inputFileBuilder.required(true);
PluginParameter<String> inputFile = inputFileBuilder.build();
```

The **available methods** to set values of the PluginParameter.Builder are

- **.description("Description")** - A short description of what the parameter is or does
- **.guiName("GUI Name")** - Specify the name for the parameter in the graphical interface, if different from the command line. (If command line name is inputFile, this defaults to "Input File")
- **.inDir() or .outDir()** - Specify that parameter is the location of an input or output directory.
- **.inFile() or .outFile()** - Specify that parameter is the location of an input or output file.
- **.range(values)** – Specify a range of acceptable values. You should use the Range class to do so:
    - **.range(Range.closed(0.0, 1.0))** – inclusive range
    - **.range(KINSHIP_METHOD.values())** - List of Enum values
    - See the Range class in the Guava library for further details (http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/Range.html)
- **.required(true/false)** – Specify if parameter is required. (Note: if set as true, you cannot define a default value for this parameter).
- **.units("unit name")** - Specify the units of the parameter ("meters", "centimorgans", etc.)
- **.dependentOnParameter(PluginParameter)** - Specify a previous Boolean parameter, that this parameter depends upon. In the GUI dialog, this parameter will only be enabled if the specified parameter is true.
- **.dependentOnParameter(PluginParameter, Object)** - Specify a previous parameter, that this parameter depends upon. In the GUI dialog, this parameter will only be enabled if the specified parameter is set to the specified value.
- **.dependentOnParameter(PluginParameter, Object[])** - Specify a previous parameter, that this parameter depends upon. In the GUI dialog, this parameter will only be enabled if the specified parameter is set to one of the specified values.
- **.genotypeTable()** - Allows user to select which component of a genotype table to use.
- **.distanceMatrix()** - Allows user to select which of the selected distance matrices to use for this parameter.
- **.taxaNameList()** - Allows user to select taxa names with searchable dialog.

- **.siteNameList()** - Allows user to select site names with searchable dialog.
- **.positionList()** - Allows user to specify a filename (.json.gz), that automatically gets imported as a PositionList

Remember to finish by calling the **.build()** method.

Step 2 – Define set/get functions in your plugin

For each PluginParameter you define, you should also define a pair of methods: one to set its value, and one to retrieve it. (These methods shouldn't include “set” and “get” in their name, but that's what they're doing, so that's how this document will refer to them.) This is so other classes can interact with the parameters. The *set* function takes the value to be set and returns the current plugin, while the *get* function just returns the value of the chosen parameter.

Technically speaking, these methods are optional because they wrap/overlap with the inherited methods setParameter(...) and getParameter(...). *However*, it's good coding practice to provide them because it makes it much easier for others to interact with your plugin from within the code base. It also means that you can call these functions from within your own plugin rather than constantly using the .value() method for each of them.

Example *set* function (from the BinaryToTextPlugin class) for the parameter myOutputFile:

```
public BinaryToTextPlugin outputFile(String value) {
    myOutputFile = new PluginParameter<>(myOutputFile, value);
    return this;
}
```

And the corresponding *get* function:

```
public String outputFile() {
    return myOutputFile.value();
}
```

Note: Automatically generating get/set functions

The GeneratePluginCode class will automatically generate all the get/set functions for your plugin (with Javadoc), assuming you’ve set it up correctly. To do so, add this main method to your plugin class and run it:

```
public static void main(String[] args) {
    GeneratePluginCode.generate(ThisPluginName.class);
}
```

(Replace *ThisPluginName* with the name of your own plugin.) All the get/set functions will be output as text to the console. You can just copy-paste them into your source code, then delete/comment out the main function.

Optionally Implement AbstractPlugin.preProcessParameters()

If any processing needs to happen before normal user input (i.e. command line flags or GUI dialog input), then put those checks in the overridden method preProcessParameters(). For example, if your plugin receives input for another Plugin.

Example

```
@Override
protected void preProcessParameters(DataSet input) {
    if (input == null) {
        throw new IllegalArgumentException("ProjectionLoadPlugin: preProcessParameters:
Please select one Genotype Table.");
    }
    List<Datum> genotypeTables = input.getDataOfType(GenotypeTable.class);
    if (genotypeTables.size() == 1) {
        myHighDensityMarkersGenotypeTable =
           (GenotypeTable) genotypeTables.get(0).getData();
    } else {
        throw new IllegalArgumentException("ProjectionLoadPlugin: preProcessParameters:
Please select one Genotype Table.");
    }
}
```

Optionally Implement AbstractPlugin.postProcessParameters()

If the value of any Parameters need checking or modifying after normal user input (i.e. command line flags or GUI dialog input), then put those checks in the overridden method postProcessParameters(). For example, if your plugin has several optional flags but requires at least one of them to be set, then this is the place to check. Or, in the example below, if no output file is given then it defaults to a standard name in the same directory as the input file. Any other validation checks should be done here also.

Example

```
@Override
protected void postProcessParameters() {
    if ((outputFile() == null) || (outputFile().length() == 0)) {
        if (inputFile() != null) {
            outputFile(Utils.getDirectory(inputFile()) + File.separator
               + "output.topm.bin");
        }
    }
    if (textOutputFormat()) {
        if (outputFile() != null) {
            outputFile(outputFile().replace(".bin", ".txt"));
        }
    }
}
```

To build the Jar file (dist/sTASSEL.jar) from the command line, run:

`ant` **or** `mvn package`

Command-line execution of a plugin:

```
./run_pipeline.pl -PluginName [parameters]
```

Call plugin this way to print its usage:

```
./run_pipeline.pl -PluginName -help
```

Example Plugin

```
package net.maizegenetics.analysis.distance;

import com.google.common.collect.Range;
import net.maizegenetics.dna.snp.GenotypeTable;
import net.maizegenetics.plugindef.AbstractPlugin;
import net.maizegenetics.plugindef.DataSet;
import net.maizegenetics.plugindef.Datum;
import net.maizegenetics.plugindef.PluginParameter;
import net.maizegenetics.taxa.distance.DistanceMatrix;
import javax.swing.*;
import java.net.URL;
import java.awt.Frame;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.apache.log4j.Logger;

/**
 * @author Terry Casstevens
 * @author Zhiwu Zhang
 * @author Peter Bradbury
 *
 */
public class KinshipPlugin extends AbstractPlugin {

    private static final Logger myLogger = Logger.getLogger(KinshipPlugin.class);

    public static enum KINSHIP_METHOD {

        Centered_IBS,
        Normalized_IBS,
        Dominance_Centered_IBS,
        Dominance_Normalized_IBS
    };

    public static enum ALGORITHM_VARIATION {

        Observed_Allele_Freq,
        Proportion_Heterozygous
    };

    private PluginParameter<KINSHIP_METHOD> myMethod = new PluginParameter.Builder<>("method",
KINSHIP_METHOD.Centered_IBS, KINSHIP_METHOD.class)
            .guiName("Kinship method")
            .range(KINSHIP_METHOD.values())
            .description("The Centered_IBS (Endelman - previously Scaled_IBS) method produces a
kinship matrix that is scaled to give a reasonable estimate of additive "
                    + "genetic variance. Uses algorithm
http://www.g3journal.org/content/2/11/1405.full.pdf Equation-13. "
                    + "The Normalized_IBS (Previously GCTA) uses the algorithm published here:
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3014363/pdf/main.pdf.")
            .build();

    private PluginParameter<Integer> myMaxAlleles = new PluginParameter.Builder<>("maxAlleles", 6,
Integer.class)
            .description("")
            .range(Range.closed(2, 6))
            .dependentOnParameter(myMethod, new Object[]{KINSHIP_METHOD.Centered_IBS,
KINSHIP_METHOD.Dominance_Centered_IBS})
            .build();

    private PluginParameter<ALGORITHM_VARIATION> myAlgorithmVariation = new
PluginParameter.Builder<>("algorithmVariation", ALGORITHM_VARIATION.Observed_Allele_Freq,
```

```
ALGORITHM_VARIATION.class)
            .description("")
            .range(ALGORITHM_VARIATION.values())
            .dependentOnParameter(myMethod, new Object[]{KINSHIP_METHOD.Dominance_Centered_IBS})
            .build();

    public KinshipPlugin(Frame parentFrame, boolean isInteractive) {
        super(parentFrame, isInteractive);
    }

    @Override
    protected void preProcessParameters(DataSet input) {
        List<Datum> alignInList = input.getDataOfType(GenotypeTable.class);
        if ((alignInList == null) || (alignInList.isEmpty())) {
            throw new IllegalArgumentException("KinshipPlugin: Nothing selected. Please select a
genotype.");
        }
    }

    @Override
    public DataSet processData(DataSet input) {

        List<Datum> alignInList = input.getDataOfType(GenotypeTable.class);

        List<Datum> result = new ArrayList<>();
        Iterator<Datum> itr = alignInList.iterator();
        while (itr.hasNext()) {

            Datum current = itr.next();
            String datasetName = current.getName();
            DistanceMatrix kin = null;

            if (current.getData() instanceof GenotypeTable) {
                GenotypeTable myGenotype = (GenotypeTable) current.getData();
                if (kinshipMethod() == KINSHIP_METHOD.Centered_IBS) {
                    kin = EndelmanDistanceMatrix.getInstance(myGenotype, maxAlleles(), this);
                } else if (kinshipMethod() == KINSHIP_METHOD.Normalized_IBS) {
                    kin = GCTADistanceMatrix.getInstance(myGenotype, this);
                } else if (kinshipMethod() == KINSHIP_METHOD.Dominance_Centered_IBS) {
                    kin = DominanceRelationshipMatrix.getInstance(myGenotype, maxAlleles(),
algorithmVariation(), this);
                } else if (kinshipMethod() == KINSHIP_METHOD.Dominance_Normalized_IBS) {
                    kin = DominanceNormalizedIBSMatrix.getInstance(myGenotype, this);
                } else {
                    throw new IllegalArgumentException("Unknown method to calculate kinship: " +
kinshipMethod());
                }
            } else {
                throw new IllegalArgumentException("Invalid selection. Can't create kinship matrix
from: " + datasetName);
            }

            if (kin != null) {
                StringBuilder comment = new StringBuilder();
                comment.append(kinshipMethod());
                if (kinshipMethod() == KINSHIP_METHOD.Dominance_Centered_IBS) {
                    comment.append("(variation: ");
                    comment.append(algorithmVariation());
                    comment.append(")");
                }
                comment.append(" matrix created from ");
                comment.append(datasetName);
                Datum ds = new Datum(kinshipMethod() + "_" + datasetName, kin, comment.toString());
                result.add(ds);
            }

        }
```

```
        return new DataSet(result, this);

    }

    @Override
    public String pluginUserManualURL() {
        return "https://bitbucket.org/tasseladmin/tassel-5-source/wiki/UserManual/Kinship/Kinship";
    }

    @Override
    public ImageIcon getIcon() {
        URL imageURL =
KinshipPlugin.class.getResource("/net/maizegenetics/analysis/images/Kin.gif");
        if (imageURL == null) {
            return null;
        } else {
            return new ImageIcon(imageURL);
        }
    }

    @Override
    public String getButtonName() {
        return "Kinship";
    }

    @Override
    public String getToolTipText() {
        return "Calculate kinship from marker data";
    }

    /**
     * Convenience method to run plugin with one return object.
     */
    public DistanceMatrix runPlugin(DataSet input) {
        return (DistanceMatrix) performFunction(input).getData(0).getData();
    }

    /**
     * The scaled_IBS method produces a kinship matrix that is scaled to give a
     * reasonable estimate of additive genetic variance. The pairwise_IBS
     * method, which is the method used by TASSEL ver.4, may result in an
     * inflated estimate of genetic variance. Either will do a good job of
     * controlling population structure in MLM. The pedigree method is used to
     * calculate a kinship matrix from a pedigree information.
     *
     * @return Kinship method
     */
    public KINSHIP_METHOD kinshipMethod() {
        return myMethod.value();
    }

    /**
     * Set Kinship method. The scaled_IBS method produces a kinship matrix that
     * is scaled to give a reasonable estimate of additive genetic variance. The
     * pairwise_IBS method, which is the method used by TASSEL ver.4, may result
     * in an inflated estimate of genetic variance. Either will do a good job of
     * controlling population structure in MLM. The pedigree method is used to
     * calculate a kinship matrix from a pedigree information.
     *
     * @param value Kinship method
     *
     * @return this plugin
     */
    public KinshipPlugin kinshipMethod(KINSHIP_METHOD value) {
        myMethod = new PluginParameter<>(myMethod, value);
        return this;
```

```
    }

    /**
     * Max Alleles
     *
     * @return Max Alleles
     */
    public Integer maxAlleles() {
        return myMaxAlleles.value();
    }

    /**
     * Set Max Alleles. Max Alleles
     *
     * @param value Max Alleles
     *
     * @return this plugin
     */
    public KinshipPlugin maxAlleles(Integer value) {
        myMaxAlleles = new PluginParameter<>(myMaxAlleles, value);
        return this;
    }

    /**
     * Algorithm Variation
     *
     * @return Algorithm Variation
     */
    public ALGORITHM_VARIATION algorithmVariation() {
        return myAlgorithmVariation.value();
    }

    /**
     * Set Algorithm Variation. Algorithm Variation
     *
     * @param value Algorithm Variation
     *
     * @return this plugin
     */
    public KinshipPlugin algorithmVariation(ALGORITHM_VARIATION value) {
        myAlgorithmVariation = new PluginParameter<>(myAlgorithmVariation, value);
        return this;
    }
}
```