

PhyloGeoRef Manual

Google Summer of Code 2010 - 2011.

Downloading and Setting Up the Environment

Using phyloGeoRef is easy. You can obtain phyloGeoRef from

<https://github.com/dapurv5/phyloGeoRef/>

Platform Java6

This can be directly opened up as a netbeans project. If you want to use the library in your own application then the best way would be to obtain the jar for the project and include it in the classpath for your project. (phyloGeoRef/dist/phyloGeoRef.jar)

If however you want to create rich kml from your phylogenetic data then you need to write a Main class.

To use the library the following jars must be placed in the classpath for the project.

1. activation.jar
2. jaxb-api.jar
3. jaxb-impl.jar
4. jsr173_1.0_api
5. JavaAPIforKml.jar
6. Junit-4.8.2.jar

These jars are placed in phyloGeoRef/dist/lib folder. To place them in the classpath Right click the project -> Properties -> Libraries -> Add JAR/Folder. Browse to the location of these jars and add them.

If you want to obtain the latest versions of these jars download the following:

- JAXB2_200xxxxx.jar (<https://jaxb.dev.java.net/>)
Unjar it using the command. `java -jar JAXB2_200xxxxx.jar`
In windows if you have WinZip installed you can unzip it simply via right clicking and choosing extract here.
The libraries that are needed in the classpath are found in jaxb-ri-200xxxxx/bin created after doing the above procedure.
- JavaAPIforKml.jar (<http://code.google.com/p/javaapiforkml/downloads/list>)

Now that we have resolved all reference problems we can move on to write some code.

Input File Formats Supported

There are two input files required by the program

- 1) Tree file
- 2) Geospatial/Metadata file

The tree file contains a phylogenetic tree. The following file types are supported: NeXML, nwk, nexus, phyloxml, tol, nhx.

The metadata file contains the location information and other metadata that might be attached to each of the taxonomic units. If the clade information is to be used then it must also be given in the metadata file itself. Any delimited text file is supported

There are a few sample files placed in the /samples folder.

See the samples/treeHypothetical/tree.xml file and samples/treeHypothetical/tree.nwk file. Both these files represent the same tree in two different formats, NeXML and newick respectively.

See the samples/treeHypothetical/test.csv file.

This is a sample metadata file in csv format. The first line must contain the tags or the column names. The node labels and latitude and longitudes can be in any order as long as you specify to the program what column corresponds to what value. Instructions to do this are below.

Writing a Main class

If you want to create kml files using the library you need to create a Main class. Let us create a Main class named Phylogeoref.

```
public class Phylogeoref{
    public static void main(String...args) throws Throwable{
        //Place the code here.
    }
}
```

Before writing the main code, we need to setup the logger. To setup the logger copy paste the following code in your class. Now the overall class looks like.

```
/**
 * Main class
 * @author apurv
 */
public class Phylogeoref {

    private final static Logger LOGGER = Logger.getLogger("nescent");

    static{
        setupLogger(); //Setup the logger at class load
    }

    /**
     * Sets up the logger.
     */
    private static void setupLogger(){
        LOGGER.setLevel(Level.ALL);
        try {
            FileHandler fhandler = new FileHandler("Logfile.txt");
            SimpleFormatter sformatter = new SimpleFormatter();
            fhandler.setFormatter(sformatter);
            LOGGER.addHandler(fhandler);

        } catch (IOException ex) {
            LOGGER.log(Level.SEVERE, ex.getMessage(), ex);
        } catch (SecurityException ex) {
            LOGGER.log(Level.SEVERE, ex.getMessage(), ex);
        }
    }

    /**
     * Utility main method, used for testing this file during development.
     * @param args
     * @throws Throwable
     */
    public static void main(String...args) throws Throwable{
```

```

        //Place the code here.
    }
}

```

The `setupLogger()` function sets up the logger. Setting up a logger is a good practice as all the logs get recorded in a log file. In our case a `Logfile.txt` gets created which contains all the logs.

Now we need to put the code in the `main()` function.

The `phyloGeoRef` pipeline consists of four essential steps.

- i. `read (nescent.phylogeoref.reader)`
- ii. `validate (nescent.phylogeoref.validator)`
- iii. `process (nescent.phylogeoref.processor)`
- iv. `write (nescent.phylogeoref.writer)`

The source code has been organized in a way to modularize each of the steps in the pipeline.

Reading

To load a file, create a new file object. If you are using a Windows machine use the following path format:

```

File treeFile = new File("samples\\treeHypothetical\\tree.nwk");
File metaFile=new File("samples\\treeHypothetical\\test.csv");

```

On Linux or Mac use a forward slash:

```

File treeFile = new File("samples/treeHypothetical/tree.nwk");
File metaFile=new File("samples/treeHypothetical/test.csv");

```

The `treeFile` may contain multiple trees but a separate metadata file needs to be prepared for each tree. To do this create an array of all the metadata files:

```

File[] metaFiles = new File[]{metaFile};

```

Now we can use the `GrandUnifiedReader` class which wraps all the reading capabilities provided by the other classes in the package.

```

GrandUnifiedReader gur = new GrandUnifiedReader();

```

Create an instance of the `GrandUnifiedReader` class.

```

gur.setTreeFile(treeFile).setMetaFile(metaFiles).setDelim(',').setCladeDiv(4)

```

Set the various parameters for it.

setDelim() sets the delimiter for the metadata file. A csv file normally uses a comma as the delimiter. If the metadata file is a text file and the delimiter character is a tab then you can specify it as

```
setDelim('\t')
```

setCladeDiv(4) is used to specify that the 4th column in the input metadata file will be used as the clade of the specie. You can also choose not to specify this in which case all the nodes will be colored in the same color.

The next thing to do is to tell the program what kind of data each column contains.

```
gur.setArgs(1,2,3);
```

args[0] = 1 , the 1st column contains the labels for the species. These uniquely define each of the species and should be the same as those used as in the input tree file.

args[1] = 2, the 2nd column contains the latitude values.

args[2] = 3, the 3rd column contains the longitude values.

Having configured the GrandUnifiedReader now invoke the following function to build the phylogenies.

```
gur.buildUnifiedPhylogeny();
```

This command does all the internal construction work done. A phylogeny and a mould map gets built which can be extracted as

```
Phylogeny phyArray[] = gur.getPhylogenyArray();  
Map mouldMapArray[] = gur.getMouldMaps();
```

If you are sure that the tree file contains only a single phylogeny then you can use.

```
Phylogeny phy = gur.getPhylogeny();  
Map mouldMap = gur.getMouldMap();
```

The mouldMap is map from the name of the taxonomic unit to the mould associated with that taxonomic unit. Each named taxonomic unit has an external mould attached with it. This mould contains additional information for the taxonomic unit, like clade and the number of observations, etc. Having obtained the phylogeny and the mould map the reading part is over.

Validating and Processing

The validation and processing steps are combined into a single class. Obtain an instance of the PhylogenyProcessor through its factory in the following manner:

```
PhylogenyProcessor processor = ProcessorFactory.getInstance(false);
```

A false value indicates that weights are not associated with the edges. That is, edge lengths are not given for the edges, or if given they are not to be used in any processing.

Now you need to phylogenify the phylogeny.

```
processor.phylogenify(phy);
```

Phylogenify means to assign location and color values to the internal nodes in the phylogeny. A false value specified before means that the midpoints will be non-weighted. A true value indicates that a weighted mean algorithm is to be used.

Writing

The last step requires writing the phylogeny into a kml. This is done via the AdvancedKmlWriter.

```
AdvancedKmlWriter kmlw = new AdvancedKmlWriter(PaintStyle.HIERARCHICAL);
```

Here we also need to specify a PaintStyle. Two kinds of PaintStyles are supported at present. HIERARCHICAL and LEVELWISE. LEVELWISE divides the kml drawn into folders for each level, whereas HIERARCHICAL draws it hierarchically.

You can create the kml or you can also prepare a compressed kmz file.

```
kmlw.createKMZ(phy, mouldMap, "mojo");
```

Here mojo is the name of the kmz file that is to be created. To create an uncompressed KML invoke the createKML() method as.

```
kmlw.createKML(phy, mouldMap, "mojo");
```

The next section deals with the internal details about the library. See `src/nescent/DemoMain.java`

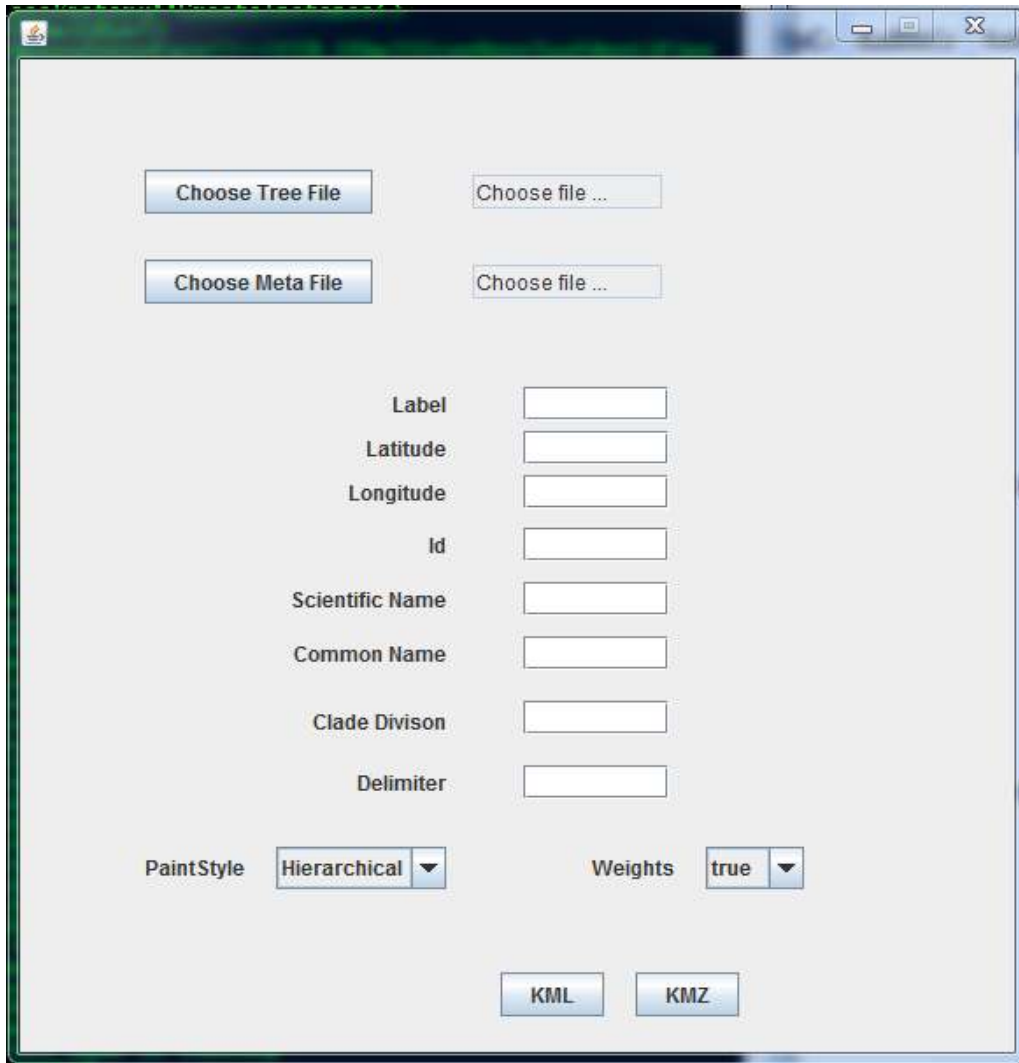
Using the simplistic GUI for easy kml generation.

It is possible to generate kmles without having to write a main class first. To do so you need the jar for the project. That jar is placed in the phyloGeoRef/dist folder.

Copy the complete dist folder to a suitable location in your computer.

```
cd dist  
java -jar phyloGeoRef.jar
```

This opens up an interactive gui. Specify the parameters and click KML to generate a kml or KMZ to generate a compressed kml file.



The screenshot shows a Java Swing window titled "phyloGeoRef.jar". The window has a light gray background and a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area contains several interactive elements:

- Two "Choose file ..." buttons, each preceded by a "Choose Tree File" and "Choose Meta File" label respectively.
- A series of text input fields for metadata, each preceded by a label: "Label", "Latitude", "Longitude", "Id", "Scientific Name", "Common Name", "Clade Divison", and "Delimiter".
- A "PaintStyle" dropdown menu currently set to "Hierarchical".
- A "Weights" dropdown menu currently set to "true".
- Two buttons at the bottom: "KML" and "KMZ".

Caveat

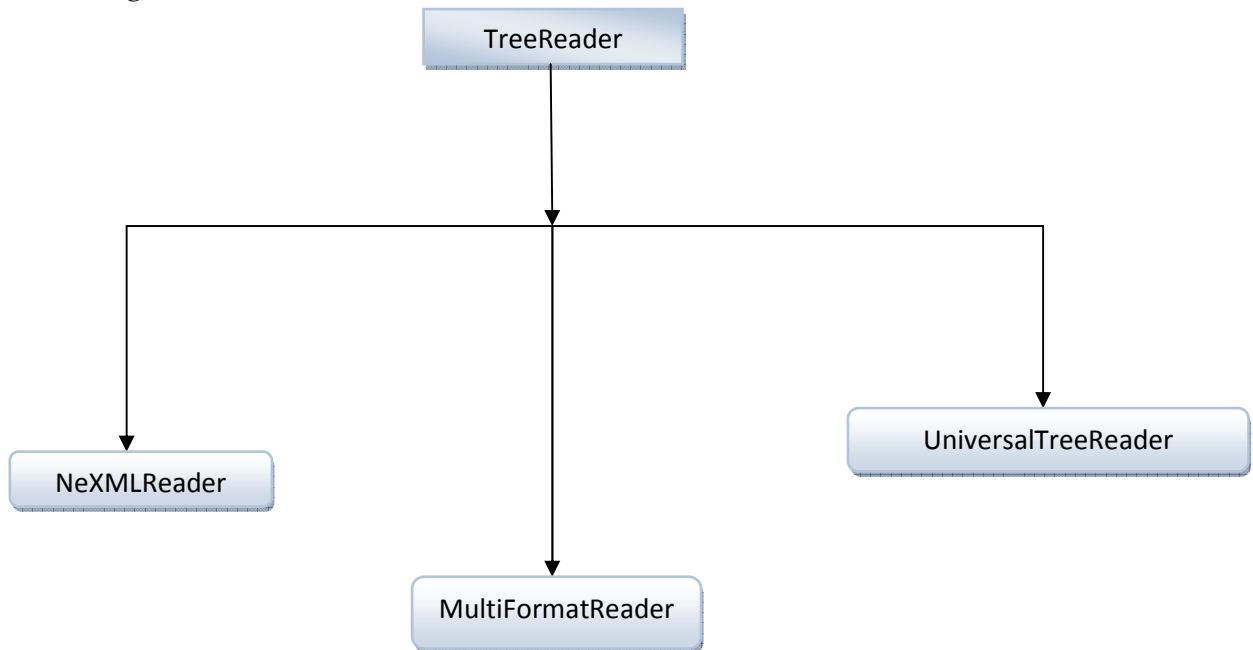
The `org.forester` library used with the project has some problems which have been overlooked. Though these errors do not raise any concerns because they are not used by the nescent library.

Here are the errors.

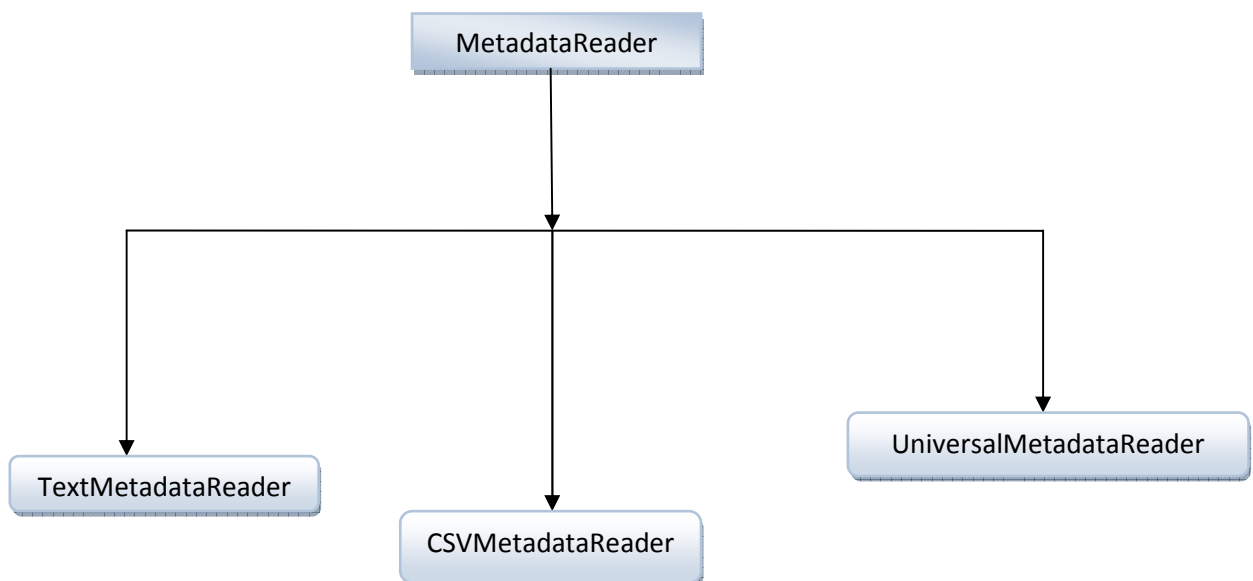
Some of the lines in the class in the file `org/forester/application/nj.java` have been commented. Search for “TODO” to find the exact lines that have been commented. These lines refer to components that do not exist in the `org.forester` library.

Understanding the Reader

This section explains how the reader component has been written and how it functions. The `nescent.phylogeoref.reader` package contains all the classes for reading input files. Here is the class diagram.



The `NeXMLReader` uses a `NeXMLEngine` behind it to construct a phylogeny from a NeXML file. Any kind of new `TreeReader` should inherit from `TreeReader` interface and the `UniversalTreeReader` must also be modified accordingly to accommodate the new type.



Any kind of new `MetadataReader` should inherit from this interface and suitable changes must be made to the `UniversalMetadataReader` to accommodate the new `MetadataReader` type.

The `GrandUnifiedReader` class combines the power of `UniversalTreeReader` and `UniversalMetadataReader` into a single class. First the `UniversalTreeReader` prepares a raw phylogeny from the tree file specified. In the second step the `GrandUnifiedReader` uses the `PhylogenyKitchen` to cook the raw phylogeny.

Adding Custom Properties

The `PhylogenyKitchen` has the following predefined properties.

```
private final static String[] PROPERTIES = {"label",
                                             "latitude",
                                             "longitude",
                                             "id",
                                             "sname",    //Scientific Name
                                             };
```

Do not change the order of these properties. This is the order that is followed when you specify the column meanings in `setArgs()`. This is the reason why `args[0]` is to be given the column number for the label, `args[1]` the column label for latitude and so on.

You can also add custom properties to it. Suppose you wish to add a custom property common name in it. Then add a variable for common name in this array.

```
private final static String[] PROPERTIES = {"label",
                                             "latitude",
                                             "longitude",
                                             "id",
                                             "sname",    //Scientific Name
                                             "cname",    //Common Name
                                             };
```

Now while specifying the `setArgs()` you need to specify the column number for `cname` also. Later to use this property go to the method

```
setNodeData(NodeData nodeData, PhylogenyMould mould)
```

Here you can simply obtain the value of that property as `get("cname")`

```
taxo.setCommonName(get("cname"));
```

P.S. Append the custom properties only at the end of the array and not in between. Also see <https://github.com/dapurv5/phyloGeoRef/wiki/Tutorial:-Using-the-GrandUnifiedReader,-adding-new-properties>.

Understanding the Validator

The validator checks for the following errors.

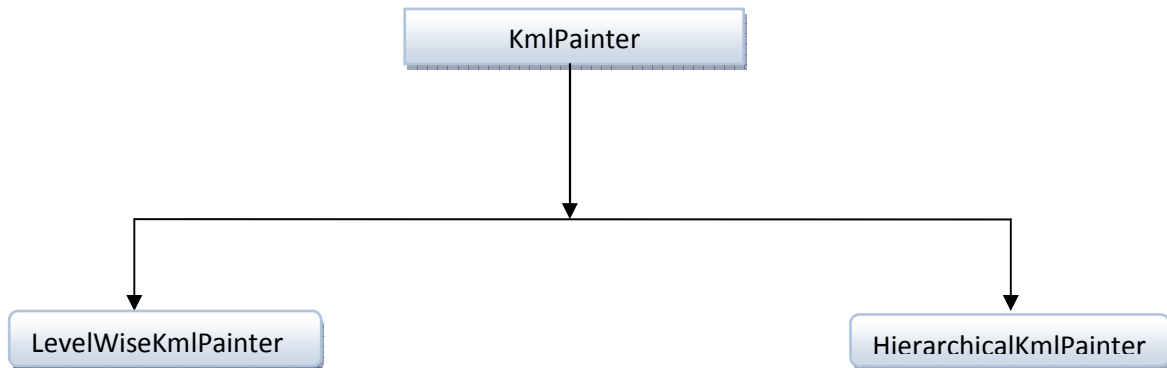
1. Tree Structure i.e. the number of children for each internal node should be at least 2
2. Latitude/Longitude values within proper bounds
3. If weighted mean algorithm is to be used then each edge should have a positive edge length associated with it.
4. Each of the external nodes must have their location information.

Understanding the Processor

The processor assigns location and color information to the internal nodes. Two kinds of processors exist. Weighted and Unweighted. The current algorithm used for finding the location of HTUs (Hypothetical Taxonomic Units) is average of latitude and longitude information. The ComputeUtility class provides the method for finding the mean longitude position. The mean is calculated in a manner so as to contain the longitude of the parent node within the bounds of the longitudes of the child nodes.

See <http://www.geomidpoint.com/calculation.html>

Understanding the Writer



LevelWiseKmlPainter draws the kml levelwise. Each level of the phylogenetic tree is drawn in a separate folder whereas a **HierarchicalKmlPainter** draws the tree hierarchically.

The various constants used in the drawing of kml are placed in `nescent.phylogeoref.utility.KmlConstants`

The **HTMLParlour** class prepares the HTML content that is placed in the balloons with each placemark.

Future Improvements

1. Using web services to get images for the species.
2. Adding Animations
3. Changing the centroid algorithm
4. Validating the complete NeXML file via HTTP.
5. Using confidence values in the kml