Tree Pruner
Technical Spec: Files, Applet Interface and Communications
Latest Draft: August 11, 2009

Overview

Tree Pruner applet, based on the LGPL Archaeopteryx "Archae" applet (http://www.phylosoft.org/archaeopteryx/), will interact with the HTTP server retrieve and store the attributes using JSON with REST.

Files / Folders

In addition to this file your package contains the following files that may be of interest to you as you read on this document

1. TreeArchaeSource/Archaeopteryx/java/appJars/TreePrunerArchae.jar : Tree Pruner based on Archae signed by flu.lanl.gov
2. TreeArchaeSource/Archaeopteryx/java/appJars/TreePrunerArchaeUnsigned.jar: Unsigned version of Tree Pruner based on Archae. This JAR will have to be signed in order for it to run from the browser.
3. TreeArchaeSource/Archaeopteryx/java/src: Source code and related Archae files. Please add all three JAR files from TreeArchaeSource/Archaeopteryx/java/jars folder to your classpath before compiling or running the code to avoid compile time errors.
4. TreeArchaeSource/Archaeopteryx/TreePrunerINSTALL/_tp_atv_configuration_file.txt : Tree Pruner – Archae configuration file.
5. TreeArchaeSource/Archaeopteryx/TreePrunerINSTALL/Example_newick_file.txt : Example newick file taken in by this applet
6. TreeArchaeSource/Archaeopteryx/TreePrunerINSTALL/Example_saved_acc_file.json : Example JSON file saved on server by the client
7. TreeArchaeSource/Archaeopteryx/TreePrunerINSTALL/exampleApplet.html : Example html applet page which can be used to run the applet by placing the necessary files in the same folder
8. TreeArchaeSource/Archaeopteryx/java/jars: Contains JSON.jar, fluorite.jar, gnujpdf.jar. These files must be included in the class path before compiling or running the TreePruner –Archae code to avoid compile time errors

Launching the JAR from HTML

TreePrunerArchae.jar or signed version of TreePrunerArchaeUnsigned.jar take in seven parameters and can be launched by using the following code snippet:

```
<Applet ARCHIVE= "TreePrunerArchae.jar" CODE="org.forester.archaeopteryx.ArchaeopteryxA.class" WIDTH=600 HEIGHT=500>
   <param name="url_of_tree_to_load" value="<relative path of newick file to load> "/>`
   <param name="config_file" value="<relative path to _tp_atv_configuration_file.txt>"/>
  <param name="filename" value="<Filename with which the accessions to be removed will be saved and retrieved from the server>"/>
   <param name="URLPrefix" value="<absolute path of URL prefix with which the accessions to be removed will be saved and retrieved from the serve> "/>
  <param name="app_type" value="0"/> <--0 = Tree Pruner – Archae, 2 = Archae -->
   <param name="saved_acc_flag" value="<true or false>"/> <-- true: if accessions were saved before; false if there is nothing saved on the server-->
  <param name ="tree_panel_tab_name" value="<This string appears on the tab of the tree panel of the Tree Pruner – Archae applet>"/>
</APPLET>
```

Newick File Format

Tree Pruner – Archae uses newick file format with the accessions of the sequences to display the tree on the applet. The accessions are used to uniquely distinguish the sequences from each other and are also passed back to the server to perform the pruning action on the server. Example format of the newick file accepted by Tree Pruner – Archae is stored in TreeArchaeSource/Archaeopteryx/TreePrunerINSTALL /Example_newick_file.txt

JSON File Format

Example of JSON file format used by Tree Pruner – Archae is stored in TreeArchaeSource/Archaeopteryx/TreePrunerINSTALL /Example_saved_acc_file.json. More details of the same can be found in the next section

Communication

Tree Pruner – Archae uses REST as architecture for communicating the above said file format back to the server and to retrieve these accessions back from the server. It uses the HTTP GET and POST request types to distinguish between getting the saved JSON string from server and sending the JSON string to server

For performing POST Tree Pruner - Archae uses the following URL, if URLPrefix passed as applet param is http://domain.com/treePruner
http://domain.com/treePruner/id.
For performing GET Tree Pruner - Archae uses the following URL, if URLPrefix passed as applet param is http://domain.com/treePruner and filename passed as applet param is 1234_56_789
http://domain.com/treePruner/id/1234_56_789

Below are the various parameters used by Tree Pruner – Archae to perform the http GET and POST methods with Java code snippets. The entire code can be found in com.lanl.application.treePruner.applet.TreePrunerCommunication for POST and in com.lanl.application.treePruner.applet.CrashRecovery for GET in src/

POST: AutoSave / Save, Commit, Discard
In order to pass the JSON file from client to server and perform autosave, save, commit and discard operations, Tree Pruner – Archae uses the HTTP POST method.

Request from Client to Server

```
...
static  String TPpostURL = AppletParams.URLprefix + "/id";
...
postURL  = new URL(TPpostURL);
HttpURLConnection
            postConn=(HttpURLConnection)postURL.openConnection();
postConn.addRequestProperty("Content-Type","text/JSON"  );
postConn.setRequestMethod("POST");
...
```
CASE 1: AutoSave/Save and Commit
Pruner action (Action), filename with which the accessions will be saved on the server – passed in as applet params (Filename) and the array of accessions to remove (Sequence Accession to Remove) are passed back in this case as a JSON format string in the payload of the packet. Example of the JSON file format is show below:

```
{"Pruner" : {
      "Action" : "<Save or Commit>",
      "Filename": "<filename>",
      "Sequence Accession to Remove" :
      [
                "ISDN12345", "ISDN5678" , ...
        ]
}}
```

CASE 2: Discard
Pruner action (Action) and the filename that will have to be removed from the server – passed in as applet params (Filename) are passed back in this case as a JSON format string in the payload of the packet. Example of the JSON file format is show below:

```
{"Pruner" : {
        "Action" : "Discard",
        "Filename": "<filename>"
}}
```

Response from Server to Client

```
...
BufferedReader rd;
Rd    =    new    BufferedReader   ( new    InputStreamReader
                (postConn.getInputStream()));
String line;
while ((line = rd.readLine()) != null) {
        returnedString += line;
}
...
```

Case 1: Commit
On receiving the packet with HTTP POST, the server performs the necessary operations based on the Action key of JSON. Then the server must respond with HTTP 200 OK or 201 Created status code along with a success message in the payload of the response, which will be simply a string that reads as “Accessions successfully removed from DB”. On receiving this, the applet will open a JOptionPane to inform the user that the task was successfully performed on the server. Failing to receive this string the applet will open a JoptionPane to tell the user that the action failed

Case 2: AutoSave / Save
On receiving the packet with HTTP POST, the server performs the necessary operations based on the Action key of the JSON. Then the server must respond with HTTP 200 OK or 201 Created status code along with a success message in the payload of the response, which will be simply a string that reads as “Accessions saved successfully”. On receiving this, the applet will dump a

statement saying that the task was successfully performed on the server. Failing to receive this the applet dump a statement saying that the action failed.

NOTE: The set on the server may need to be locked to avoid the saved accessions on the server from becoming stale.

Case 3: Discard
On receiving the packet with HTTP POST, the server performs the necessary operations based on the Action key of the JSON. Then the server must respond with HTTP 200 OK or 201 Created status code along with a success message in the payload of the response, which will be simply a string that reads as "Accessions successfully discarded". On receiving this, the applet will dump a statement saying that the task was successfully performed on the server. Failing to receive this the applet dump a statement saying that the action failed

And finally the connection is closed.
```
postConn.disconnect();
```

GET: Crash Recovery / Resuming an existing Session
In order to get the JSON file from the server and perform crash recovery or resume an existing session of the Pruner, Tree Pruner – Archae uses the HTTP GET method.

Request from Client to Server

```
...
String TPgetURL = AppletParams.URLprefix + 
            "/id/"+AppletParams.filename;
savedAccURL = new URL(TPgetURL);
HttpURLConnection crashRecConn = (HttpURLConnection) 
            savedAccURL.openConnection();
crashRecConn.setRequestMethod("GET");
...
```

The client sets the request method as GET and tries to connect to the GET URL with an empty payload

Response from Server to Client

```
...
BufferedReader rd;
rd = new BufferedReader(new InputStreamReader
            (crashRecConn.getInputStream()));
String line;
```

```
while ((line = rd.readLine()) != null) {
     TPgetJSON += line;
}
...
```

On receiving the GET request, the server uses the filename in the URL to fetch the JSON format string on the server Server then responds with a packet that has HTTP 200 OK or 201 Created status code, the content type of the packet - text/JSON and JSON string saved on the server as the payload.

And finally the connection is closed.

```
crashRecConn.disconnect();
```

If something is wrong with the JSON format in the string got back from the payload of the response, JSONTokener will throw a JSONException on the client.

Enjoy the product!