

---

# **cartogratree Documentation**

*Release 7.x-0.1-dev*

**Nic Herndon**

**Oct 03, 2020**



---

## Table of contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Project members . . . . .	1
<b>2</b>	<b>Administrators</b>	<b>3</b>
2.1	Installation . . . . .	3
2.2	Configuration . . . . .	4
<b>3</b>	<b>Database setup</b>	<b>7</b>
3.1	Chado . . . . .	7
3.2	Trees . . . . .	7
3.3	Layers . . . . .	10
3.4	Sessions . . . . .	12
<b>4</b>	<b>Users</b>	<b>13</b>
4.1	The Left Panel . . . . .	13
4.2	Select dataset sources to include . . . . .	15
4.3	Select and filter trees . . . . .	15
4.4	Select environmental data . . . . .	19
4.5	Get location information . . . . .	21
4.6	Navigation Controls . . . . .	21
4.7	Saving and loading saved searches and config . . . . .	21
4.8	Analyze data . . . . .	23
<b>5</b>	<b>Developer Documentation</b>	<b>25</b>
5.1	About . . . . .	25
5.2	Coding style specifications . . . . .	25
5.3	Project Structure . . . . .	26
5.4	Mapbox . . . . .	27
5.5	TreeSnap . . . . .	27
5.6	BIEN . . . . .	27
5.7	Adding a new Tree Dataset source . . . . .	28
5.8	Adding Geoserver Tileset layers . . . . .	28
5.9	Customizing icons on the map that represent the points . . . . .	28
5.10	Filtering using a Query Builder . . . . .	29
5.11	Updating the NodeJS API . . . . .	31
5.12	Extracting tree data . . . . .	31
5.13	Extracting layer data . . . . .	33

5.14	Adding and Removing layers from the map . . . . .	33
5.15	Adding layer filters for environmental layers . . . . .	34
5.16	Adding layer legend box HTML . . . . .	34
5.17	Generating analysis tables and passing to Sambada . . . . .	34
5.18	Saving and loading sessions . . . . .	35
<b>6</b>	<b>CartograTree API Reference</b>	<b>37</b>
6.1	CONFIGURATION . . . . .	37
6.2	ADMIN API FUNCTIONS . . . . .	37
6.3	REMOVE TREEGENES TREES . . . . .	37
6.4	TREES RELOAD . . . . .	38
6.5	TREES RELOAD STATUS . . . . .	38
6.6	CACHE CLEAR . . . . .	38
6.7	CONSOLIDATE DRYAD TREES FROM TREEGENES . . . . .	39
6.8	CONSOLIDATE TREESNAP SUBKINGDOMS . . . . .	39
6.9	CONSOLIDATE TREESNAP FAMILIES . . . . .	39
6.10	RELOAD TREESNAP . . . . .	40
6.11	FORCE RELOAD TREESNAP . . . . .	40
6.12	FORCE RELOAD BIENv4 TREES . . . . .	40
6.13	FORCE RELOAD BIENv4 ENDANGERED TAXA . . . . .	41
6.14	NORMAL USER API FUNCTIONS . . . . .	41
6.15	GET TREES . . . . .	41
6.16	GET TREES BASED ON QUERY . . . . .	42
6.17	GET TREES BASED ON SOURCE . . . . .	44
6.18	GET TREE BASIC INFORMATION . . . . .	45
6.19	GET TREE PHENOTYPES . . . . .	46
6.20	GET TREE GENOTYPES . . . . .	47
6.21	GET TREE MARKERS . . . . .	48
6.22	GET TREE PUBLICATIONS . . . . .	49
6.23	GET TREE TREESNAP COMPLETE . . . . .	51
6.24	GET SESSION DATA . . . . .	52
6.25	GET USER SESSION . . . . .	53
6.26	GET ALL USER SESSIONS . . . . .	54

Forest trees are long-lived and immobile individuals that serve as ideal models to assess population structure and adaptation to the environment. Despite the availability of comprehensive data, the researchers who study them are challenged to integrate data describing genotype, phenotype, and the environment. Towards this goal, the web application CartograTree was designed and implemented as an open repository and open-source analytic web-based framework for all three. This framework allows query and analysis in a map-based interface to primarily enable association mapping, ecological genomics, and landscape genomics, through high performance computing. The high performance computing and hosted applications are connected to the user queried data via web services. Ontologies implemented for sequence, phenotype, and environmental metrics facilitate these transactions.

### 1.1 Project members

- Nic Herndon
- Taylor Falk
- Emily Grau
- Sean Beuhler
- Peter Richter
- Risharde Ramnath
- Ronald Santos
- Jill Wegrzyn



## 2.1 Installation

Note: The following instructions assume that **CartograTree** will be installed on a Linux server.

### 2.1.1 CartograTree Drupal module

CartograTree is a module for **Drupal 7** CMS, dependent on the **Tripal 7.x-3.x**, **TripalGalaxy 7.x-1.x** modules, and **GeoServer v2.11.2**. Installation instructions for **Drupal 7**, **Tripal**, and **GeoServer** are available online. **TripalGalaxy** can be installed by cloning it from GitHub, in the `sites/all/modules` directory of Drupal, and then enabling it with Drupal shell (`drush`).

```
$ git clone https://github.com/tripal/tripal_galaxy.git
$ drush en tripal_galaxy
```

Once these dependencies are installed, you can proceed with installing the **CartograTree** module, in the `sites/all/modules` directory of Drupal.

```
$ git clone https://gitlab.com/TreeGenes/CartograTree.git
## create cartogratree directory and copy the module contents in it
$ mkdir cartogratree
$ mv CartograTree/drupal_module/* cartogratree/
## remove the other CartograTree components
$ rm -rf CartograTree/
## enable the module
$ drush en cartogratree
```

## 2.1.2 Library dependencies

### 2.1.3 Mapbox

**CartograTree** uses **Mapbox**, a mapping software, for creation of its base map and to host some of its layers as tilesets. **CartograTree** uses the **MapboxGL JS** library to add layers and data to the map.

The MapBox GL JS API reference can be found here: <https://docs.mapbox.com/mapbox-gl-js/api/>

To use Mapbox, you first need to create an account and get an access token.

### 2.1.4 CartograTree API

A final dependency for the **CartograTree** Drupal module is the **CartograTree API**. This in turn depends on **NodeJs v8.11.2**, **NPM v5.6.0**, and **PM2**. This API also uses the modules **mem-cache**, **cors**, **pg**, **body-parser**, and **response-time**. To install the **CartograTree API** follow these steps:

```
## create a directory for this API
$ mkdir -p cartogratree/api
## copy the API files into in
$ git clone https://gitlab.com/TreeGenes/CartograTree.git
$ mv CartograTree/NodeJsAPI/* cartogratree/api/
## remove the other CartograTree components
$ rm -rf CartograTree/
## replace the values for host, port, database, user, and password in query.js
## install the express module
$ cd cartogratree/api
$ npm install express pg --save
## install PM2 and run it as a service
$ sudo npm install pm2@latest -g
$ sudo pm2 startup systemd
## confirm that it's running
$ sudo systemctl status pm2-root.service
## start CartograTree API and add it to PM2's process list
$ pm2 start cartogratree_api
## to [stop/start/restart/get info] run:
$ pm2 [stop|start|restart|info] cartogratree_api
```

## 2.2 Configuration

Go to your Drupal website and navigate to `admin/cartogratree/settings`. From this page you can set up the URLs for the GIS and API servers (`admin/cartogratree/settings/servers/edit`), **addeditdelete** groups and subgroups for organizing the layers, and **addeditdelete** **access permissions** for layers. For environmental layers, you can also get the values at the locations of the trees.

### 2.2.1 Groups

For groups, two values are required: **Group name** and **Group rank**. The **Group name** will be displayed on **CartograTree** page, and the **Group rank** determines the location in the list for this group (the lower the rank the higher in the list).



## 2.2.2 Subgroups

For subgroups, only one value is required: **Subgroup name**. Once created, a subgroup can be added to one or more groups. The **Subgroup name** will be displayed on CartograTree page, based on the rank assigned in the group.

## 2.2.3 Layers

Once the groups and subgroups are defined, the layers can be added to CartograTree. The layer page prompts for the following values:

- **Human-readable name**: this is the name shown to the CartograTree users. Make sure it is descriptive, and uniquely identifies the layer.
- **Machine name**: this is the name used by the GIS server.
- **URL**: the location from where this layer was downloaded.
- **Trees layer[yes/no]**: tree layers are shown on top of the environmental layers.
- **Group name** and **Subgroup name**, discussed above.
- **Layer rank**: the order in which layers within a subgroup are listed. If from worldclim for example, then the layer ranks will range from 1-12 corresponding with the months January - December.
- **Fields**: which fields from this layer are *exposed* to the users (described below).

The machine name is defined by you when you add and publish the layer to geoserver. For adding mapbox layers log in to your mapbox account and go to: <https://studio.mapbox.com/tilesets/>. Convert your layer to one of the accepted formats, add it as a tileset, then go to <https://studio.mapbox.com/>, select the style of your base map or create a new one. Then click on the 'Add Layer' button at the top left of the screen and select your tileset as the source.

### Fields

- **Field name**: this is the name returned by the GIS server.
- **Display name**: this is the name shown to the users.
- **Filter[yes/no]**: whether to allow the users to filter the data based on this field's values.
- **Filter type[continuous/discrete]**: the types of values present in this field.
- **Precision used with continuous values**
- **Mask value**: value returned by layer that should be replaced.
- **Mask display**: text shown to user for masked values.
- **Pop-up** whether this field should be shown in the pop-up window, when the user clicks on the map.
- **Field rank**: the order in which fields within a layer are listed.

### Layer permissions

Using this form, administrators can control which users or Drupal roles can access each layer.

### Get layer values

Using this form, administrators can extract the values from an environmental layer at the locations with geo-referenced trees, and then save these values in the database.



CartograTree and its API depend on several tables and materialized views. Their schemas are shown below. If the images are too small and hard to see, right click -> 'View Image' will give you the full image resolution.

### 3.1 Chado

Chado is a db schema optimized for genomic data. CartograTree makes use of several chado tables to store and receive its data. Read about it more here: [http://www.gmod.org/wiki/Chado\\_Manual](http://www.gmod.org/wiki/Chado_Manual)

### 3.2 Trees

#### 3.2.1 ct\_trees\_all\_view

This is the main view used to populate the map with tree data. It's a view that is created from joining multiple tables to allow for greater read access speed.

Column	Type	Storage
uniquename	character varying(255)	extended
family	character varying(255)	extended
genus	character varying(255)	extended
species	character varying(255)	extended
subkingdom	character varying(255)	extended
latitude	double precision	plain
longitude	double precision	plain
coordinate_type	integer	plain
source_id	integer	plain
icon_type	integer	plain
tree_num	integer	plain
phenotype_name	character varying(1024)	extended
pato_name	character varying(1024)	extended
po_name	character varying(1024)	extended
cvterm_name	character varying(1024)	extended
marker_type	character varying(1024)	extended
title	text	extended
author	text	extended
accession	character varying(1024)	extended

### 3.2.2 ct\_trees

This is the table which is used by ct\_trees\_all\_view to cross reference the trees with their genomic and phenotypic data. It contains all the trees.

Column	Type	Modifiers	Storage	Stats target	Description
uniquename	character varying(255)	not null	extended		
genus	character varying(255)		extended		
species	character varying(255)		extended		
subkingdom	character varying(255)		extended		
family	character varying(255)		extended		
latitude	double precision		plain		
longitude	double precision		plain		
coordinate_type	integer		plain		
source_id	integer		plain		
icon_type	integer		plain		
tree_num	integer	not null default nextval('ct_trees_tree_num_seq'::regclass)	plain		

Indexes:  
 "ct\_trees\_pkey" PRIMARY KEY, btree (uniquename)  
 has OIDs: no

### 3.2.3 chado.ct\_view

This is the main view that houses the trees submitted through TPPS. It uses join conditions on tables such as chado.stock and chado.stockprop to build each record.

Column	Type	Storage
uniquename	text	extended
genus	character varying(255)	extended
species	character varying(255)	extended
subkingdom	text	extended
family	text	extended
latitude	double precision	plain
longitude	double precision	plain
coordinate_type	character varying(20)	extended
geometry	geometry	main

### 3.2.4 chado.new\_genotype\_view

This view contains tree genotype records.

Column	Type	Storage
tree_acc	text	extended
marker_name	text	extended
description	text	extended
marker_type	character varying(1024)	extended
study_type	text	extended
quality_score	text	extended
marker_technology	text	extended
assembly_type	unknown	plain
scaffold	unknown	plain
position	unknown	plain
original_name	unknown	plain

### 3.2.5 chado.new\_pheno\_view

This view contains tree phenotype records.

Column	Type	Storage
tree_acc	text	extended
name	text	extended
username	text	extended
phenotype_name	character varying(1024)	extended
value	text	extended
units	text	extended
po_name	character varying(1024)	extended
po_accession	character varying(1024)	extended
pato_name	character varying(1024)	extended
pato_accession	character varying(1024)	extended
cvterm_name	character varying(1024)	extended
cvterm_accession	character varying(1024)	extended

### 3.2.6 chado.plusgeno\_view

This view contains tree publication records.

Column	Type	Storage
project_id	bigint	plain
accession	character varying(1024)	extended
pub_id	bigint	plain
title	text	extended
year	character varying(255)	extended
author	text	extended
species	text	extended
tree_count	bigint	plain
phen_count	numeric	main
ontology_ids	text	extended
phenotypes_assessed	bigint	plain
gen_count	bigint	plain
markers	text	extended
study_type	text	extended

## 3.3 Layers

The layers tables are important for the administration side because they are what decide which layers are available and useable to the user.

### 3.3.1 cartogratree\_layers

Column	Type	Modifiers	Storage	Stats target	Description
layer_id	integer	not null default nextval('cartogratree_layers_layer_id_seq'::regclass)	plain		Primary Key
title	character varying(512)		extended		Human-readable name for the layer.
name	character varying(256)		extended		Machine name for the layer.
url	character varying(256)		extended		The URL for the provider of the layer.
group_id	integer		plain		Accordion group ID for side navigation menu.
subgroup_id	integer		plain		Accordion subgroup ID for side navigation menu.
layer_rank	integer		plain		Layer rank for side navigation menu.
trees_layer	integer		plain		
layer_type	character(6)		extended		
layer_host	integer	default 0	plain		Indicates the type of layer: vector, or raster.

Indexes:  
 "cartogratree\_layers\_pkey" PRIMARY KEY, btree (layer\_id)  
 Has OIDs: no

### 3.3.2 cartogratree\_layer\_permissions

Column	Type	Modifiers	Storage	Stats target	Description
layer_permissions_id	integer	not null default nextval('cartogratree_layer_permissions_layer_permissions_id_seq'::regclass)	plain		Primary Key
layer_id	integer	not null	plain		Links to the layers table.
rid	integer		plain		The ROLE ID of a role to provide access to.
uid	integer		plain		The USER ID of a user to provide access to.

Indexes:  
 "cartogratree\_layer\_permissions\_pkey" PRIMARY KEY, btree (layer\_permissions\_id)  
 Has OIDs: no

### 3.3.3 cartogratree\_fields

Column	Type	Modifiers	Storage	Stats target	Description
field_id	integer	not null default nextval('cartogratree_fields_field_id_seq'::regclass)	plain		Primary Key
layer_id	integer		plain		Parent layer of this field.
field_name	character varying(256)		extended		Name returned by layer.
display_name	character varying(256)		extended		Name shown to user.
filter	integer	default 0	plain		Allow users to filter data by this field (1), or not (0).
filter_type	character varying(10)		extended		Type of filter to use with this field: radio-button, checkbox, slider, or elastic search (radio, check, slider, or elastic, respectively).
field_values	text		extended		Range for slider (e.g., 0-20), and semi-colon sperated list f or radio and check.
precision	integer	default 2	plain		Precision used with range values.
mask_value	character varying(256)		extended		Value returned by layer that should be masked.
mask_display	character varying(256)		extended		Text shown to user for masked values.
pop_up	integer	default 0	plain		Show this field in maps pop-up (1), or not (0).
data_table	integer	default 0	plain		Show this field in data table (1), or not (0).
field_rank	integer		plain		Order in which this field is shown.
filter_category	character varying(256)		extended		

Indexes:  
 "cartogratree\_fields\_pkey" PRIMARY KEY, btree (field\_id)  
 Has OIDs: no

### 3.3.4 cartogratree\_groups

Column	Type	Modifiers	Storage	Stats target	Description
group_id	integer	not null default nextval('cartogratree_groups_group_id_seq'::regclass)	plain		Primary Key
group_name	character varying(256)		extended		Accordion group name for side navigation menu.
group_rank	integer		plain		Group rank for side navigation menu.

Indexes:  
 "cartogratree\_groups\_pkey" PRIMARY KEY, btree (group\_id)  
 Has OIDs: no

### 3.3.5 cartogratree\_subgroups

Column	Type	Modifiers	Storage	Stats target	Description
subgroup_id	integer	not null default nextval('cartogratree_subgroups_subgroup_id_seq'::regclass)	plain		Primary Key
subgroup_name	character varying(256)		extended		Accordion subgroup name for side navigation menu.

Indexes:  
 "cartogratree\_subgroups\_pkey" PRIMARY KEY, btree (subgroup\_id)  
 Has OIDs: no

### 3.3.6 cartogratree\_groups\_subgroups

Column	Type	Modifiers	Storage	Stats target	Description
group_id	integer	not null	plain		FK from cartogratree_groups
subgroup_id	integer	not null	plain		FK from cartogratree_subgroups
subgroup_rank	integer	not null	plain		Subgroup rank for side navigation menu.

Indexes:  
 "cartogratree\_groups\_subgroups\_pkey" PRIMARY KEY, btree (group\_id, subgroup\_id, subgroup\_rank)  
 Has OIDs: no

## 3.4 Sessions

Session tables must also be created to store logged in user and anonymous user sessions.

### 3.4.1 ct\_sessions

Column	Type	Modifiers	Storage	Stats target	Description
session_id	character varying(32)	not null	extended		
created_at	timestamp without time zone	not null default now()	plain		
updated_at	timestamp without time zone		plain		
zoom	double precision		plain		
bearing	double precision		plain		
pitch	double precision		plain		
included_trees	jsonb		extended		
selected_filters	jsonb		extended		
selected_layers	jsonb		extended		
center	jsonb		extended		

Indexes:  
 "ct\_sessions\_pkey" PRIMARY KEY, btree (session\_id)

Referenced by:  
 TABLE "ct\_sessions\_user" CONSTRAINT "ct\_sessions\_user\_session\_id\_fkey" FOREIGN KEY (session\_id) REFERENCES ct\_sessions(session\_id)

has OIDs: no

### 3.4.2 ct\_user\_sessions

Column	Type	Modifiers	Storage	Stats target	Description
user_id	character varying(64)	not null	extended		
title	character varying(128)	not null	extended		
comments	character varying(512)		extended		
session_id	character varying(32)	not null	extended		

Indexes:  
 "ct\_sessions\_user\_pkey" PRIMARY KEY, btree (user\_id, session\_id)

Foreign-key constraints:  
 "ct\_sessions\_user\_session\_id\_fkey" FOREIGN KEY (session\_id) REFERENCES ct\_sessions(session\_id)

has OIDs: no

Once all these tables and views have been created in your PostgreSQL database, then go to the API directory of the project and with your favorite text editor edit query.js

Input the necessary values for the following configuration fields in query.js

```
const conf = {
  host: <host>,
  port: <port>,
  database: <database_name>,
  user: <username>,
  password: <password>,
  ...
};
```



### 4.1 The Left Panel





The action panel to interact with the map and trees is located to the left of the screen. At the top lies the button to expand the panel to show the environmental layers panel. After that there are two buttons: the RESET MAP button, which resets the map to its original state on load and the CLOSE TREE VIEW button, which closes the tree view that is opened when a tree is clicked. The section after that shows the map summary, which has statistics for the data selected and shown on the map. At the bottom of the left panel is a collapse button, which hides the majority of the action panel increasing the size of the map.

**Map Options**

ENVIRONMENTAL LAYERS +

RESET MAP      CLOSE TREE VIEW

**Map Summary**

 Number of Trees	36818
 Number of Species	137
 Number of Publications	71
 Number of Layers	0

**Tree Dataset Sources**

**Filters**

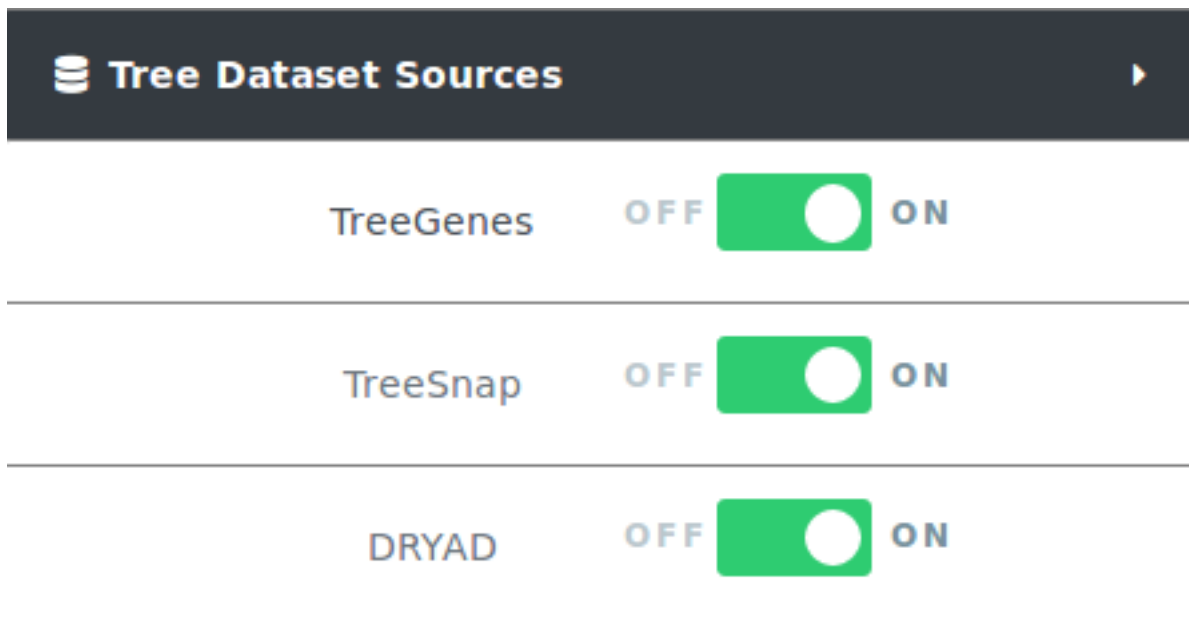
**<< Collapse**

## 4.2 Select dataset sources to include

CartograTree enables the analysis of trees from three data sources: TreeGenes, TreeSnap, and DataDryad. The user can choose which trees from which datasets they want on the map, or any combination of them. When applying filters only trees from selected datasets are effected.

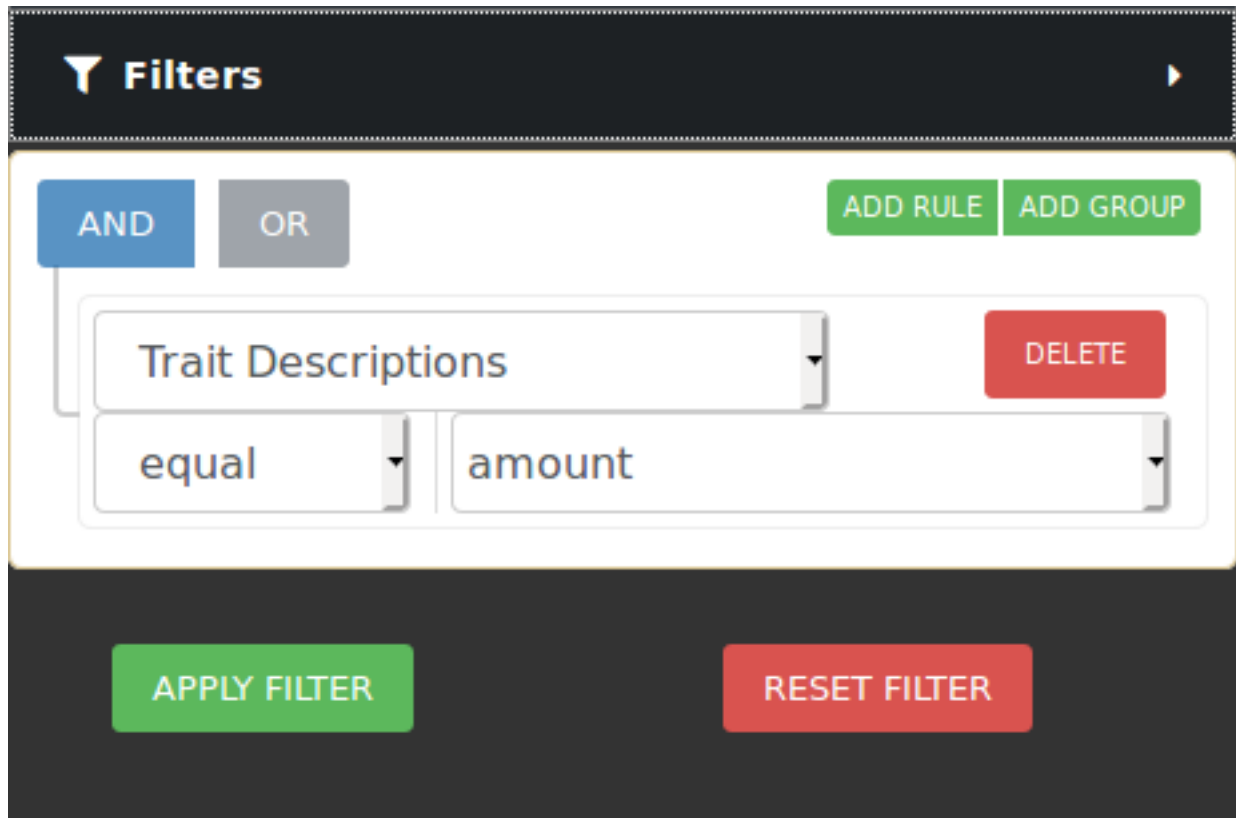
To contribute and submit your own data for TreeGenes, you can use TPPS: <https://treegenesdb.org/tpps>. This is a module that streamlines the submission process for plant related studies. The documentation for TPPS can be found here: <https://tpps.readthedocs.io/>

You can also contribute to TreeSnap by downloading the app and submitting pictures of the trees that you find: <https://treesnap.org/>.



## 4.3 Select and filter trees

Users can select from the left pane the trees to be included in an analysis. CartograTree has tens of thousands of trees, and it will only continue to grow. To only look at the trees of interest to you, you can filter the trees on the map by taxonomy, publication, genotype, and/or phenotype.



If you want to narrow your search even more, you can make use of the combination filters offered by making use of the AND/OR switches and the ADD RULE and ADD GROUP buttons. Resetting the filter removes all the currently active filters and shows all trees available on the map again.

The screenshot displays a filter configuration interface. At the top, there are two buttons: 'AND' (highlighted in blue) and 'OR' (grey). To the right are 'ADD RULE' and 'ADD GROUP' buttons. Below this is a filter rule for 'Trait Descriptions' with a 'DELETE' button. The rule is set to 'equal' and 'amount'. A second filter group is shown below, also with 'AND' (blue) and 'OR' (grey) buttons, and 'ADD RULE', 'ADD GROUP', and 'DELETE' buttons. This second group contains a single filter rule with a 'DELETE' button. The filter value is currently '-----'.

The operation that is currently selected for each group is colored blue. The selection condition available for each property is EQUAL and NOT EQUAL. After clicking Apply Filters, the map should move to the area of interest where the filtered trees are located.

Hovering over a tree icon reveals how many trees there are at that location. Clicking on a tree icon opens a map overlay located at the top right of the map. This map overlay shows information about the currently selected tree and any images that are present for that tree. Clicking on the ADDITIONAL INFO button creates a popup which shows the publication, markers and phenotypes recorded for the tree. At the bottom is a list of the trees located at this area. You can click on any of the tree IDs on the list to view information for that tree. There is also an ADD TREE button which puts the tree in the list of trees that will be part of the analysis.

TGDR035-151167  
36.83 Long | -102.96 Lat

Image 1

Pinaceae  
Pinus ponderosa

Plant group: gymnosperm  
Source: TreeGenes  
Coordinate Type: Exact

ADD TREE

- TGDR035-151167
- TGDR035-151151
- TGDR035-151166
- TGDR035-151152
- TGDR035-151170

### 4.3.1 Filtering by passing parameters to URL

You can also filter the trees on load of the map by adding additional parameters to the URL. The base URL for CartograTree is: <https://treegenesdb.org/cartogratree> If a valid session has been provided then the URL is: [https://treegenesdb.org/cartogratree?session\\_id=<session\\_id>](https://treegenesdb.org/cartogratree?session_id=<session_id>)

The additional parameters that you can pass include TGDR study accession, species, genus, and family. Passing multiple of these parameters will result in a conjunction of the options. An example filter by passing URL arguments is: <https://treegenesdb.org/cartogratree?accession=TGDR002> <https://treegenesdb.org/cartogratree?species=Abies%20alba>

## 4.4 Select environmental data

Users can also select from the left pane, the environmental values to be included in an analysis, by turning environmental layers on or off. The environmental layers are layered on top of each other, with the latest layer placed on top of previously selected layers. Clicking on any random point on the map with at least one active layer will show the environmental values for that location.


◀ **Environmental Layers** ☰

**WorldClim v.2 (WorldClim)** ▶

**Precipitation (WorldClim v2)** ▼

Precipitation  OFF  ON

January

Opacity  100%

---

**Temperature (WorldClim v2)** ▼

---

**Solar Radiation (WorldClim v2)** ▼

---

**Wind Speed (WorldClim v2)** ▼

---

**Water Vapor (WorldClim v2)** ▼

**Major Soil Types (Conservation Biology Institute)** ▼

**Species Range Maps (USFS)** ▼

**Land Cover (USGS)** ▼

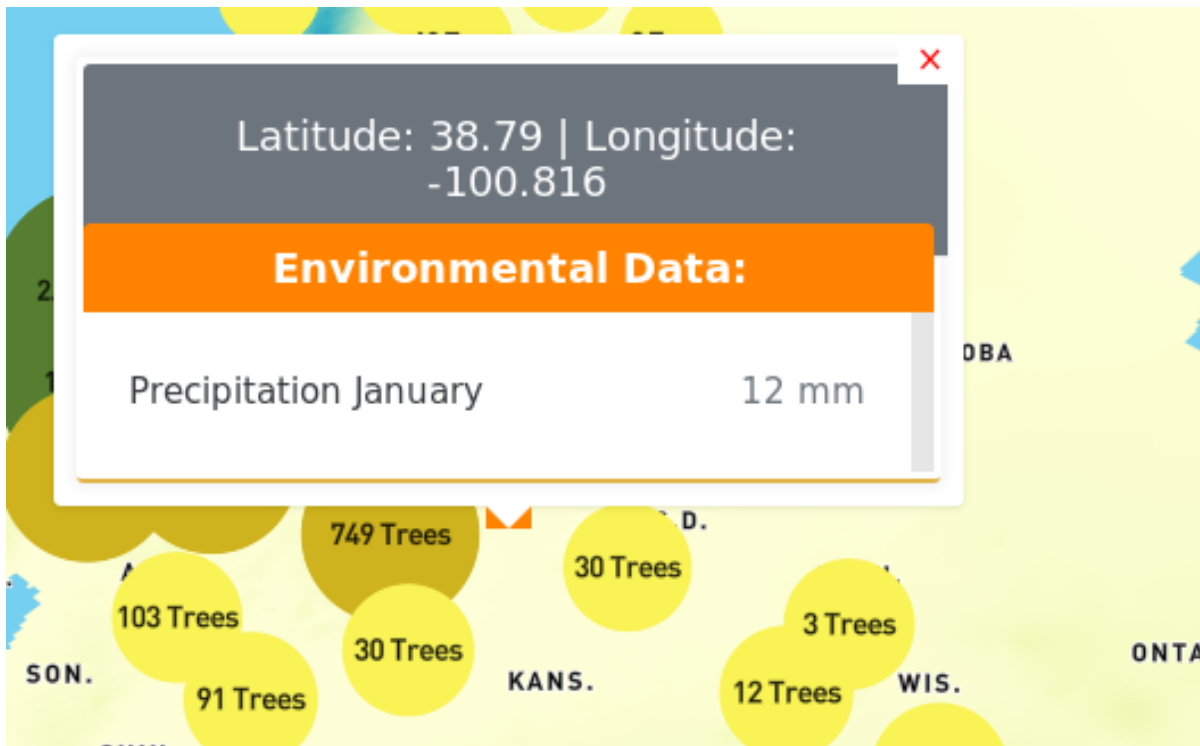
**PET and Aridity (CGIARCSI)** ▼



Some environmental layers contain additional filters which will also be seen under the on/off toggle button. For example, range maps may have color options that allow you to color each range map with a different color as a way to differentiate between multiple selected layers. Some environmental layers contain a year range slider filter which allows you to filter between a starting year and ending year. For those that can filter by year, you will see a range slider in which the left select represents the starting year and while the right select represents the ending year. Adjusting these filters will filter the data in realtime.

## 4.5 Get location information

Users can display the values at a particular location on the map, by clicking on the map at that location. The pop-up window, will display the trees present at that location (if there are any), and the values from the environmental layers selected.



At zoomed out levels the trees are clustered and are represented by circles. The color and size of the circle indicates how many trees there are in the cluster. The user can click on a cluster to zoom in and view the individual trees.

## 4.6 Navigation Controls

You can use arrow keys to move across the zoom, and the scroll wheel to zoom in. Double clicking on any point in the map will also zoom you in. There also exists navigation controls at the top right of the map which gives you control of the zoom level, the bearing and the pitch of the map.

## 4.7 Saving and loading saved searches and config

The filters applied, layers activated, trees chosen and current state of the map can be saved to go back to at a later date by clicking the SAVE SESSION button located at the bottom of the left panel. If the save is successful, a popup will

appear with a link with a session\_id supplied for CartograTree which you can follow at anytime to go back to this state of the map. Verified users that are logged in to their account can also save a session to their account with a title and comment.

## Save Session



Title

Comments

CLOSE

SAVE SESSION

## Save Session



Session successfully saved. Copy and paste the link below to go back to this session, or view all your saved sessions from the dropdown link located below your name at the top right portion of the navbar.

[https://tgwebdev.cam.uchc.edu/cartogratree?session\\_id=b8cea7bdcad203ff1b3327525e4ec848](https://tgwebdev.cam.uchc.edu/cartogratree?session_id=b8cea7bdcad203ff1b3327525e4ec848)

CLOSE

## 4.8 Analyze data

Logged in users have permission to perform landscape genomics analysis. The user can select the dataset to be used, select/deselect specific trees, what publications to include, and which environmental variables to be sent for analysis. In the final stage before confirmation, they will be shown histograms related to the genotypic data associated with each tree and the user can select and filter out data they would like to exclude. The analysis will be performed using **Samβada** and the user will be notified of the results after the analysis is done.

### 4.8.1 Galaxy Workflows

### 4.8.2 How do I select trees?

When you click on a tree icon on the map, a detailed view pops up to the right of the screen. On the bottom of this map overlay, there is an “Add All” button to add all the trees at this location for analysis, and an “Add Tree” button next to the tree ids to add individual trees.

### 4.8.3 How do I select environmental data?

You can view environmental layers using the environmental layers panel, then clicking on any point in the map when any of the environmental layers active shows the value at that location for the layers active. For analysis, you can select the environmental variables you want based on the layers active.

### 4.8.4 How long does it take?

Depends on the size of the dataset you selected. Can be anywhere from a couple hours to days.

### 4.8.5 How do I know when it's done?

You'll receive an email notification.

### 4.8.6 Example Input

Name	ENV1	ENV2	ENV3
ID1	46	89	99
ID2	10	2	77
ID1	46	89	99
ID2	10	2	77
ID1	46	89	99
ID2	10	2	77

### 4.8.7 Example Output

## 5.1 About

CartograTree was built with CSS3, HTML5, JavaScript, PHP 7.0.29, and PSQl 9.2.15. The libraries and frameworks used include: Mapbox GL JS 1.0.0, Bootstrap 4.0.0, and JQuery 2.2.4. You should also be familiar with Git to commit and push changes. Understanding Mapbox and Drupal is a good starting point before diving deep into the project.

If you want to read about the motivations and future plans for the project read this paper: [https://docs.google.com/document/d/1y4-KwPzZfSC6bzHZlhFI627cLPwfz0H3ogZUb\\_XQeMs](https://docs.google.com/document/d/1y4-KwPzZfSC6bzHZlhFI627cLPwfz0H3ogZUb_XQeMs)

Before anything else, make sure you have sudo access in the server where CartograTree is running and admin privileges in the Drupal website where CartograTree is being developed. Although not required these extra privileges will make development much easier. You should have a GitLab account to be able to contribute to the codebase, Geoserver and Mapbox accounts to view the layers, a TreeSnap account with a valid API key to request TreeSnap data, and an account in the Drupal website where CartograTree is installed to be able to log in and test the changes.

## 5.2 Coding style specifications

It helps to maintain consistency in code to make maintainability of the codebase much easier. Some coding guidelines for developing CartograTree:

- Tabs are 4 spaces, indent using tabs not spaces
- Starting curly brace should be in the same line, while ending curly brace in their own line
- Don't add in-line comments for the sake of adding comments, only do so if an explanation is necessary. The names of functions or variables should be enough in some cases, if they were named appropriately.
- HTML ids and classes delimited by '-'
- In PHP only use single quotes unless necessary
- Try to use Tripal API functions instead of Drupal ones
- In JavaScript single and double quotes are interchangeable, but try to keep it consistent

- Only use ternary operators if it could fit in a single line or less
- Class names should be capitalized
- **Variables**
  - Should describe what the variable is
  - No one letter variable names unless used for loops or temporary variables
  - Should be short and sweet
  - For PHP and NodeJS we use underscore, and for front-end JavaScript we use camelCase
  - Try to avoid using global variables
- **Functions**
  - Don't repeat yourself
  - Don't make functions too long
  - Ideally a function should only do one thing
  - Try to be as descriptive in the name as possible, but not too long that it becomes a pain to type or refer back to
  - For PHP and NodeJS we use underscore, and for front-end JavaScript we use camelCase
  - Minimize vertical distance for functions such that functions that use each other are close to each other in the top-down view of the program
- Any tokens or API keys should not be exposed to the public and ideally should be stored in a configuration file or as environmental variables

## 5.3 Project Structure

```
- drupal_module
  - cartogratree.info
  - cartogratree.install
  - js
    - main.js
    - map.js
  - styling
    - main.css
  - includes
    - cartogratree_admin_get_environmental_values.js
    - cartogratree.admin.inc
    - cartogratree.user.inc
  - theme
    - templates
      - page--cartogratree.tpl.php
      - resources_imgs
      - trees_imgs
      - ui_icons_imgs
    - cartogratree_theme.inc
  - cartogratree.module
- NodeJSAPI
  - cartogratree_api.js
  - query.js
  - routes.js
```

(continues on next page)

(continued from previous page)

```
- controllers
  - treeController.js
  - userController.js
  - oldController.js
- package.json
- README.md
- galaxy_workflows
- LICENSE
- README.md
- CHANGELOG.md
```

The cartogratree-page.tpl.php contains the basic HTML layout of the page. The map.js file contains the majority of the code for controlling the map and any map interactions. The cartogratree.module is the heart of the Drupal program and controls what happens on page load.

Modifying any changes to the files will usually allow you to view the effects on refresh of the page, but in the event it doesn't try clearing the cache by typing the following command in the terminal:

```
drush cc all
```

## 5.4 Mapbox

Mapbox is a platform for hosting and editing map styles. It also offers libraries and frameworks for creating web-based mapping applications. CartograTree uses Mapbox GL JS to display its base map and serve some of its layers. Cartogratree's base map along with the composited layers can be found here: [https://api.mapbox.com/styles/v1/snkb/cjrgce15209bi2spi9f2ddvch.html?fresh=true&title=true&access\\_token=pk.eyJ1Ijoic25rYiIsImEiOiJjanFtZXpkbmkzc2cyM3hsYjJ3dDhtYnp5In0.suo54ZuchrNHspBirQ8drA#4.05/40.54/-96.66](https://api.mapbox.com/styles/v1/snkb/cjrgce15209bi2spi9f2ddvch.html?fresh=true&title=true&access_token=pk.eyJ1Ijoic25rYiIsImEiOiJjanFtZXpkbmkzc2cyM3hsYjJ3dDhtYnp5In0.suo54ZuchrNHspBirQ8drA#4.05/40.54/-96.66)

In case you're stuck on something Mapbox related a good place to start is the documentation, which you can find here: <https://docs.mapbox.com/mapbox-gl-js/api/>

## 5.5 TreeSnap

CartograTree works in collaboration with TreeSnap and their mobile application which relies on citizen scientists to submit trees they encounter in nature. To use the TreeSnap API you must first create a TreeSnap account and get a bearer token which will be used to authenticate API requests. Create a TreeSnap account then go to <https://treesnap.org/developer> to generate your tokens for use. Documentation for the TreeSnap API and what data you can request can be found at: <https://github.com/statonlab/Treesnap-website/wiki/Observations-Web-Service>

## 5.6 BIEN

CartograTree also works in collaboration with the Botanical Information and Ecology Network's dataset. The BIEN working group has been working since 2008 toward bringing together disparate networks of botanical researchers and integrating global botanical observation data. This observational data is pulled into Cartogratree for further research and analysis. Please have a look at Cartogratree's API function list for more information on related BIENv4 features. Currently Cartogratree supports two main observation sub datasets from BIENv4 - tree observations and endangered taxa.

## 5.7 Adding a new Tree Dataset source

The tree dataset sources on the map are loaded as GeoJSON files received from an API endpoint. GeoJSON is a format for encoding a variety of geographic data structures. The GeoJSON returned must be a list of features with valid longitude and latitude coordinates and a unique id. All of the trees are stored in a single table where each record has a <source\_id> column. CartograTree currently supports three sources, where TreeGenes = 0, TreeSnap = 1, DataDryad = 2 and BIEN = 3 for source ids. Currently the dataset layer, which encompasses the three sources mentioned beforehand, is composed of three layers. The first layer, which is the outermost layer, is the cluster text layer. This displays how many trees there are in this cluster. The 2nd top most level layer is the cluster layer itself, which is just a circle symbol on the map with varying sizes and colors. The last layer is for the individual points which are represented by tree icons and only appear when the user is zoomed in.

An example GeoJSON feature collection array

```
{ "type": "FeatureCollection",
  "features": [
    { "type": "Feature", "properties": { "icon_type": 1, "id": "ABAL0001" }, "geometry": {
    ↪ "type": "Point", "coordinates": [10.8377888, 45.9715095] } },
    { "type": "Feature", "properties": { "icon_type": 1, "id": "ABAL0002" }, "geometry": {
    ↪ "type": "Point", "coordinates": [10.8380944, 45.9715108] } }
  ]
}
```

The <id> in the properties object identifies a single tree. The <icon\_type> property signifies which icon the tree will have displayed on the map to represent it. Currently there are six icon types, where exact/approximate refers to the confidence level of the location of the tree.

```
0 = Angiosperm exact
1 = Gymnosperm exact
2 = Angiosperm approximate
3 = Gymnosperm approximate
4 = TreeSnap angiosperm
5 = TreeSnap gymnosperm
```

To read more about GeoJSON and how to optimize it for large datasets: <https://docs.mapbox.com/help/troubleshooting/working-with-large-geojson-data/>

## 5.8 Adding Geoserver Tileset layers

Our latest version of Cartogratree uses a mixture of both geoJSON layers and Geoserver tileset layers to display datasets. For example, the new BIEN layer is a Geoserver tileset layer. In order to add Geoserver tileset layers, it is important to name it and also append ‘\_geoserver\_tileset’ within the Javascript code particularly for the global variable datasetKey which holds the corresponding source\_ids of each layer and the datasetGeoserverLayerConfig which holds the raw Geoserver layer name for the dataset and other required settings. It is also important to edit the theme file page-cartogratree.tpl.php which contains the dataset toggle buttons and set an id for the new dataset in the form of <name>\_geoserver\_tileset-data.

## 5.9 Customizing icons on the map that represent the points

The different icon\_types differentiate between approximate and exact coordinates of the georeferenced trees, the plant group that the tree belongs to, as well as the data source. TreeSnap’s trees implement some sort of coordinate obfuscation as some of the trees recorded are on private property.



To add your own icons, first the icons must be chosen. You can either choose from some of mapbox's [icons](#) and add them to your base map using mapbox studio, or add your own images to the project directory and linking them yourself.

#### Adding your own icons to mapbox

```
map.loadImage(<image_path>,
  function (error, <image_var_name>) {
    map.addImage(<image_name_to_reference>, <image_var_name>);
  }
)
```

To match the trees with their icon a mapbox matching expression is used.

```
[
  "match",
  ["get", <icon_property_name>],
  <icon_property_value>,
  <image_name_to_reference>,
  <icon_property_value>,
  <image_name_to_reference_2>,
  <default_image_name_to_reference>,
];
```

If keeping the trees intact, then you can simply replace the images located in the `theme/templates/trees_imgs` directory with your new tree images.

## 5.10 Filtering using a Query Builder

The `main.js` file located in the `/js` directory contains the starter code for the filter options available to the user. The `ruleBasic` variable below indicates what is the first filter option shown to the user on load. The `<condition>` denotes the starting selected condition for this query. The `<condition>` parameter can either be 'AND' or 'OR'.

```
var rulesBasic = {
  condition: 'AND',
  rules: [
    {
      id: 'family',
      operator: 'equal',
    },
  ],
};
```

The `filtersList` variable contains all of the filters declared and that are available to the user right now. Each filter is composed of several properties. The `<id>` indicates the field of the selection and must be unique, the `<label>` is the name shown to the user, `<input>` is the type of form being used. The `<values>` is the available options for this select dropdown, it's composed of key/value pairs. This is dynamically populated on page load. The `<operators>` is the array of clauses that is available for this option.

```
var filtersList = [
  {
    id: 'family',
    label: 'Family',
    type: 'string',
    input: 'select',
```

(continues on next page)

(continued from previous page)

```

        values: {},
        operators: ['equal', 'not_equal']
    },
    ...
]

```

To get the filters chosen by the user we select the element that contains the filter form and call the queryBuilder function: `$('#builder').queryBuilder('getRules')`; This returns a json object, which we pass to the API and the API parses it, generates a sql query, then returns back the result of the query.

An example json that can result from a filter selection

```

{
  "condition": "AND",
  "rules": [
    {
      "id": "family",
      "field": "family",
      "type": "string",
      "input": "select",
      "operator": "equal",
      "value": "Pinaceae"
    },
    {
      "condition": "AND",
      "rules": [
        {
          "id": "genus",
          "field": "genus",
          "type": "string",
          "input": "select",
          "operator": "equal",
          "value": "Pinus"
        },
        {
          "id": "marker_type",
          "field": "marker_type",
          "type": "string",
          "input": "select",
          "operator": "equal",
          "value": "SNP"
        }
      ]
    }
  ]
}

```

### 5.10.1 Parsing

The filters chosen by the user can be as deep as the user wants, so a recursive function that simulates a depth first search approach is used to parse the resulting json object. We go through each filter, and if we encounter a filter that is an object we go one level deeper. We use parenthesis to separate between groups of filters.

The resulting query from the example json above would look like:

```
SELECT * FROM some_table WHERE (family = 'Pinaceae') AND (genus = 'Pinus' AND marker_
↳type = 'SNP');
```

## 5.11 Updating the NodeJS API

The NodeJS express API is composed of four major parts. The `cartogratree_api.js` is the main file, declares the necessary dependency modules and starts everything. Then there's the `routes.js` file which defines all the current available API endpoints. Adding a new endpoint is fairly trivial. Then there's the `query.js` file which defines the connection to the database and what gives the API access and ability to get data from the tables.

```
app.<operation>('/cartogratree/api/<version>/<path>', <controller>.<function_name>);
```

Here the `<operation>` refers to GET, POST, PUT, etc operations. The `<version>` refers to the API's current version, right now CartograTree is at version 2. This version number should remain a whole number just by convention.

Lastly, there are the controllers which receive the requests handled by the routes. The controllers must be first imported into the `routes.js` file to be able to refer to them. The general format for a controller function that handles an API call is:

```
exports.<function_name> = function(req, res, next) {
  do stuff
  res.status(<status_code>).json(<some_result>);
  return;
}
```

To see your changes take effect, you need to restart the API. If using `systemd` to manage the api:

The command: `sudo systemctl restart cartogratree_api` restarts the API.

To see the current status of the API do: `sudo systemctl status cartogratree_api`

If you are using the `node` command:

The command: `node cartogratree_api &` will start the server for the API in the background

To stop it: Find the process id of the the program using `ps aux | grep node` and kill it using `kill -9 <node_process_id>`

Using `node` to run the server is recommended as it allows you to see a more detailed error message.

To view debugging output: `node cartogratree_api` on the console.

## 5.12 Extracting tree data

When a user clicks on a point, we can grab the features within that click point.

```
map.on('click', function (e) {
  //create a bounding box around the clicked point, to be used to query for_
  ↳features/trees around the bbox
  var treesBbox = [
    [e.point.x, e.point.y],
    [e.point.x, e.point.y]
  ];

  var features = map.queryRenderedFeatures(treesBbox, {
```

(continues on next page)

(continued from previous page)

```
    layers: [<layer_names>]
  });
```

Using the features, we can find the properties of the points at the clicked location and from the properties find the id of the trees. Then we make an API call where we pass in the tree\_id to get the information we need and want to display. For TreeSnap trees, we need to also communicate with their API to get the data and images we need.

This is what a typical result looks like when querying features of a clicked point

```
[
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -122.97992706298828,
        40.19228260049829
      ]
    },
    "type": "Feature",
    "properties": {
      "id": "FS0085",
      "icon_type": 1
    },
    "layer": {
      "id": "tree-points",
      "type": "symbol",
      "source": "trees",
      "filter": [
        "!",
        [
          "has",
          "point_count"
        ]
      ]
    },
    "layout": {
      "icon-image": [
        "match",
        [
          "get",
          "icon_type"
        ]
      ],
      0,
      "angiosperm_ex",
      1,
      "gymnosperm_ex",
      2,
      "angiosperm_app",
      3,
      "gymnosperm_app",
      4,
      "treesnap_angio",
      5,
      "treesnap_gymno",
      "angiosperm_ex"
    ],
    "icon-size": 0.04,
    "icon-allow-overlap": true
  }
]
```

(continues on next page)

(continued from previous page)

```

    }
  },
  "source": "trees",
  "state": {}
},
...
]

```

Here the `tree_id` of the first point in the clicked area is “FS0085”. After making a call to the API and passing in this `tree_id` this should be the response

```

{
  "uniquename": "FS0085",
  "genus": "Pinus",
  "species": "Pinus balfouriana",
  "subkingdom": "gymnosperm",
  "family": "Pinaceae",
  "latitude": 40.19228333333333, "longitude": -122.97993333333333,
  "coordinate_type": 0,
  "source_id": 0,
  "icon_type": 1,
  "tree_num": 1486
}

```

## 5.13 Extracting layer data

Mapbox layers are composited into the map, so we can get the environmental data at a clicked point just by looking at the properties of the layers around the clicked point. For geoserver layers we must get the data using `GetFeatureInfo`. For more information: <https://docs.geoserver.org/latest/en/user/tutorials/GetFeatureInfo/index.html>

Getting the environmental data around a clicked point is done by making an ajax call to an endpoint similar to this one:

```
https://tgwebdev.cam.uhc.edu/geoserver/wms?SERVICE=WMS&VERSION=1%2E3%2E0&REQUEST=GetFeatureInfo&LAYERS=[X1, Y1], [X2, Y2]]
```

Where `<layer_name>` is the name of the layer that takes a form of a string BBOX is the bounding box and takes the form of an array

## 5.14 Adding and Removing layers from the map

The layers of CartograTree are hosted in both geoserver and mapbox. Adding raster layers to the base map on mapbox doesn't seem to give good results, at least not from the layers we have, and the most success seems to come from converting the layer to GeoJSON using something like QGIS and generating tile sets based on the GeoJSON. To add a mapbox layer to the map that has already been composited to the base map, we merely need to change the layer's visibility settings.

Mapbox has a library for converting GeoJSON data to vector tilesets: <https://github.com/mapbox/tippecanoe>

To learn more about combining layers: <https://docs.mapbox.com/help/troubleshooting/mapbox-gl-js-performance/#combine-layers>.

## 5.15 Adding layer filters for environmental layers

We've introduced 2 available environmental layer filters which use different concepts.

1. Color filters
2. Year filters.

To enable color filters for environmental layers, you can visit the Cartogratree Admin UI at `admin/cartogratree/settings` and either create or edit a layer. You can enable the color filter by checking the checkbox labeled 'Allow user to select/change colors via the map interface?'. This will add a color picker user interface option when a user selects an environmental layer and allow users to change the color of this layer in realtime.

To enable year filters for environmental layers, visit the Cartogratree Admin UI and add or edit a layer. You will see a checkbox labeled 'Allow user to filter by year range via the map interface?'. Make sure to select this checkbox and also ensure that you set the additional fields such as Year range geoserver parameter, Year range starting year and Year range ending year. The year range geoserver parameter is the parameter used to filter the year range and is usually specific to the fields embedded in a layer (within Geoserver). Currently this filter is only compatible with year data that is stored as integers and will not work properly if the field is saved as a string in Geoserver. The start and ending year field limits the UI slider to only select years between these years.

## 5.16 Adding layer legend box HTML

You can also provide custom information for each environmental layer which will then be available to users by utilizing the Layer Legend HTML field. You can access this field by going to the Cartogratree Admin UI at `admin/cartogratree/settings` and adding or editing a layer. You should be able to see the Layer Legend HTML text area. You can enter plain text or HTML text in this text area. Users will be able to view this information by clicking on the blue info (i) icon when they enable a layer.

## 5.17 Generating analysis tables and passing to Sambada

Marker type and environmental value tables are created, written to a file and submitted to a galaxy workflow to perform analysis. Sambada is the framework used for landscape genomics analysis. To generate the marker types table, which will be entirely composed of SNP, we get all the markers from the trees selected that are of the SNP type. We store all the available SNP markers into a set, then go through each of the SNPs in the set and cross reference with the SNP markers that the current tree has. If the current SNP from the entire dataset chosen does not exist for the current tree, then we set the cell value to 'NaN' otherwise we set it to whatever corresponding value the tree has based on homogenous and heterogenous from reference. We perform some further array operations on the resulting 2D array which simulates the table to generate a chart for it using D3js. The user can further filter the SNP table using the generated chart filter options.

For the environmental value table we collect all the `tree_ids` of the selected trees, and create a mapping of coordinate location to array of `tree_ids` in that location. Then we go through all the keys of the mapping, which will be of an array of coordinates, and extract the environmental values based on the activated layers.

We write the final arrays generated into a CSV file, and send to the galaxy Sambada workflow using the Tripal Galaxy API. To check for results and whether an analysis has been completed, the Tripal Galaxy API functions were used.

## 5.18 Saving and loading sessions

Based on the `session_id` provided in the url, if the `session_id` is valid then all session related data from that session is loaded to the map on page load. The `session_data` is requested from the API and takes the form of a JSON object. Logged in users have access to all their saved sessions and can load or delete any one of them as they wish. Saving sessions is entirely under the user's control, if the user is not logged in then the session data is saved, and if the user is logged in then the user's session identification parameters such as title and comments are also saved. The complex items such as layers and filters applied are saved as jsonb data types, while the other values are saved as more primitive sql data types.





---

## CartograTree API Reference

---

CartograTree communicates with its own API to send and request data. To gain access to this API, first create a TreeGenes account and generate an API key. The API contains both normal user level API functions as well as administrative API functions. The API key is required for normal user-level API usage such as accessing tree information while the asup key is needed to perform administrative functions such as clearing the cache.

### 6.1 CONFIGURATION

The latest version of Cartogratree API (NodeJS) uses configuration settings within the config.js file.

The most relevant configuration variable in relation to the Cartogratree API is the asup key which is an abbreviation for api\_superuser\_password.

You must set this password appropriately in order to perform administrative level API functions such as remove\_treegenes\_trees, reload\_trees and reload\_trees\_status.

### 6.2 ADMIN API FUNCTIONS

The functions below require the ?asup= get variable in order to execute.

### 6.3 REMOVE TREEGENES TREES

Removes TreeGenes trees from the public.ct\_trees table. This function is used before the TREES RELOAD function.

Type of Request: GET

URL: [http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/remove\\_treegenes\\_trees?asup=<password>](http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/remove_treegenes_trees?asup=<password>)

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.4 TREES RELOAD

Regenerates the views that contain tree information that are used by Cartogratree API. This function answers to the browser but performs multiple tasks in the background. Regenerating the trees can take several minutes or even hours to complete depending on the size of your tree database. Most of the tree\_reload function performs nested SQL/Database manipulations and this is why the function can take so long - since multiple views and tables are dependent on each other. The following are generated in the order listed:

- chado.new\_genotype\_view
- chado.plusgenotype\_view
- chado.new\_phenotype\_view
- chado.ct\_view
- ct\_trees\_all\_view

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/trees_reload?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.5 TREES RELOAD STATUS

Returns whether the TREE RELOAD function is currently regenerating the views. This can be used to determine if the task is ongoing/running or not running. You can reload this page multiple times to get the current status.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/reload_status?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.6 CACHE CLEAR

Most SQL results are stored within the cache for usually 24 hours. Especially during development, this is not usually ideal - this is where the cache\_clear function is useful. It clears all memory records and this assures you that the information you are retrieving from the Cartogratree UI is valid and fresh.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/clear?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.7 CONSOLIDATE DRYAD TREES FROM TREEGENES

Converts TPPS submitted trees that are categorized as DRYAD studies into the correct database table `ct_trees`. It updates the `source_id` to 2 which is the DRYAD category. This must be done periodically and is usually called with a cron every 12 hours on production environments. This allows Cartogratree to properly display these trees as DRYAD trees. It also affects the DRYAD dataset filter.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/dryad/consolidate_dryad_trees_from_treegenes?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.8 CONSOLIDATE TREESNAP SUBKINGDOMS

TreeSnap trees are imported from the TREESNAP database (remotely). During the import, any trees with their subkingdom value as NULL in the `ct_trees` table is cross referenced with the `organismprop` table in an attempt to retrieve a valid subkingdom and them updated.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/treesnap/consolidate_subkingdoms?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.9 CONSOLIDATE TREESNAP FAMILIES

TreeSnap trees /date are imported from the TREESNAP database (via the remote API). During the import, any trees with their family value as NULL in the `ct_trees` table is cross referenced with the `organismprop` table in an attempt to retrieve a valid family and then updated.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/treesnap/consolidate_families?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.10 RELOAD TREESNAP

TreeSnap trees / data are imported from the TREESNAP database (via the remote API). From time to time, when TreeSnap updates it's remote database, it is important for Cartogratree to pull the latest updates. This API function call will do this, continuing where it left off instead of reimporting the whole database.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/treesnap/reload?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.11 FORCE RELOAD TREESNAP

TreeSnap trees / data are imported from the TREESNAP database (via the remote API). This is mostly used for major debugging. This function will pull the entire TREESNAP database into Cartogratree.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/treesnap/force_reload?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.12 FORCE RELOAD BIENV4 TREES

This function imports BIEN tree data from the publicly accessible BIENV4 database. This function uses unique TreeGenes genus groups and query the BIENV4's massive dataset for related trees. It does not import the entire BIENV4 dataset but rather a subset from the distinct genus groups from TreeGenes. This data is imported into the public.ct\_trees table with the source\_id set as 3.

Due to the size of this dataset (millions of records returned), nodejs may need more memory for this function to successfully complete. To avoid out of memory errors and incomplete imports, you can tweak NodeJS to run with a higher amount of memory (once available on server). Example command to do so (assigning 4 GB of memory to NodeJS): `node --max-old-space-size=4096 cartogratree_api.js`

**IMPORTANT:** Like most force reload functions, this force reload function will delete all data from the table before importing data. Please remember this is resource intensive (NodeJS memory intensive) and also database server (IO

writes and CPU) and can take a day or more to fully complete. It is not designed to be used on a daily basis for refreshing data.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/bienv4/trees/force_reload?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.13 FORCE RELOAD BIENV4 ENDANGERED TAXA

This function imports BIEN endangered taxa data from the publicly accessible BIENV4 database. This function downloads the entire dataset into Cartogratree for display purposes. The data is imported into the `public.ct_bien_endangered_taxa` table.

**IMPORTANT:** Like most force reload functions, this force reload function will delete all data from the table before importing data.

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/bienv4/endangered_taxa/force_reload?asup=<password>`

Query string parameters:

Name	Type	Description
asup	string	API Super User Password

## 6.14 NORMAL USER API FUNCTIONS

The functions below only require the `?api_key=` get variable in order to execute.

## 6.15 GET TREES

Gets all the trees currently curated by TreeGenes and other sources

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees?api_key=<api_key>`

Query string parameters:

Name	Type	Description
api_key	string	Your API key

### 6.15.1 Example Response

```
[{"type": "Feature", "properties": {"icon_type": 1, "id": "ABAL0001"}, "geometry": {"type": "Point", "coordinates": [10.8377888, 45.9715095]}}, {"type": "Feature", "properties": {"icon_type": 1, "id": "ABAL0002"}, "geometry": {"type": "Point", "coordinates": [10.8380944, 45.9715108]}}
...
]
```

### 6.15.2 Response Fields

Field	Type	Description
icon_type	integer	The type of icon for the feature
id	string	The id of the feature
geometry.type	string	The type of geometry
geometry.coordinates	array of doubles	Coordinates [long, lat]

## 6.16 GET TREES BASED ON QUERY

Gets all the trees based on the active datasets on the map and the filter created by the user

Type of Request: POST

URL: [http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/q?api\\_key=<api\\_key>](http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/q?api_key=<api_key>)

Query string parameters:

Name	Type	Description
api_key	string	Your API key
query	object	The query object derived from query builder
active_sources	array of strings	The active datasets on the map

Sample parameters passed

```
{
  "query": {
    "condition": "AND",
    "rules": [
      {
        "id": "family",
        "field": "family",
        "type": "string",
        "input": "select",
        "operator": "equal",
        "value": "Anacardiaceae"
      }
    ]
  },
  "active_sources": [
    "0",
    "1",
    "2"
  ]
}
```

(continues on next page)

(continued from previous page)

```
]
}
```

### 6.16.1 Example POST request using ajax

```
$.ajax({
  url: 'https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/q?api_key=<api_key>
  ↪',
  type: 'POST',
  async: false,
  contentType: 'application/json',
  data: JSON.stringify(jsonData),
  success: function(result){ ... },
  error: ...
});
```

### 6.16.2 Example Response

```
{
  "features": [
    {
      "type": "Feature",
      "properties": {
        "id": "TGDR045-161613",
        "icon_type": 0
      },
      "geometry": {
        "type": "point",
        "coordinates": [
          -54.36032,
          -31.01048
        ]
      }
    },
    ...
    {
      "type": "Feature",
      "properties": {
        "id": "TGDR045-161614",
        "icon_type": 0
      },
      "geometry": {
        "type": "point",
        "coordinates": [
          -54.35471,
          -31.01187
        ]
      }
    },
  ],
  "center": [
    -56.10413,
```

(continues on next page)

(continued from previous page)

```

    -30.2429
  ],
  "num_species": 1,
  "num_pubs": 1,
  "num_trees": 278
}

```

### 6.16.3 Response Fields

Field	Type	Description
features	array of objects	A feature list of points
center	array of doubles	The coordinate where most points are located
num_species	integer	Number of species resulting from this query
num_pubs	integer	Number of publications resulting from this query
num_trees	integer	Number of trees resulting from this query

## 6.17 GET TREES BASED ON SOURCE

Gets all the trees based on their dataset source

Type of Request: GET

URL: [http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/source?api\\_key=<api\\_key>&source\\_id=<source\\_id>](http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/source?api_key=<api_key>&source_id=<source_id>)

Query string parameters

Name	Type	Description
api_key	string	Your API key
source_id	integer	The id of the source dataset

### 6.17.1 Example GET request

[https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/source?api\\_key=<api\\_key>&source\\_id=1&source\\_id=2](https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/trees/source?api_key=<api_key>&source_id=1&source_id=2)

### 6.17.2 Example Response

```

[{"type":"Feature","properties":{"icon_type":1,"id":"ABAL0001"},"geometry":{"type":
↪"Point","coordinates":[10.8377888,45.9715095]}}},
 {"type":"Feature","properties":{"icon_type":1,"id":"ABAL0002"},"geometry":{"type":
↪"Point","coordinates":[10.8380944,45.9715108]}}
  ...
]

```



### 6.17.3 Response Fields

Field	Type	Description
icon_type	integer	The type of icon for the feature
id	string	The id of the feature
geometry.type	string	The type of geometry
geometry.coordinates	array of doubles	Coordinates [long, lat]

## 6.18 GET TREE BASIC INFORMATION

Get the basic information of the tree

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/tree?api_key=<api_key>&tree_id=<tree_id>`

Query string parameters:

Name	Type	Description
api_key	string	Your API key
tree_id	string	The unique id of the individual tree

### 6.18.1 Example GET request using ajax

```
$.ajax({
  url: 'https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/tree?api_key=<api_key>&tree_id=ABAL0001',
  type: 'GET',
  contentType: 'application/json',
  data: JSON.stringify(jsonData),
  success: function(result){ ... },
  error: ...
});
```

### 6.18.2 Example Response

```
{
  "uniquename": "ABAL0001",
  "genus": "Abies",
  "species": "Abies alba",
  "subkingdom": "gymnosperm",
  "family": "Pinaceae",
  "latitude": 45.9715095,
  "longitude": 10.8377888,
  "coordinate_type": 0,
  "source_id": 0,
  "icon_type": 1,
  "tree_num": 1
}
```

### 6.18.3 Response Fields

Field	Type	Description
uniquename	string	The id of the tree
genus	string	The genus of the tree
species	string	The species of the tree
subkingdom	string	The plant group the tree belongs to
family	string	The family of the tree
latitude	double	The latitude coordinate of the tree
longitude	double	The longitude coordinate of the tree
coordinate_type	integer	The type of coordinate for the tree recorded
source_id	integer	The dataset source that the tree originates from
icon_type	integer	The type of icon that represents the tree on the map
tree_num	integer	The number of the tree

## 6.19 GET TREE PHENOTYPES

Get the phenotype information of the tree

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/phenotypes?api_key=<api_key>&tree_id=<tree_id>`

Query string parameters:

Name	Type	Description
api_key	string	Your API key
tree_id	string	The unique id of the individual tree

### 6.19.1 Example GET request using ajax

```
$.ajax({
  url: 'https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/phenotypes?api_key=<api_
  ↪key>&tree_id=TGDR069-190112',
  type: 'GET',
  contentType: 'application/json',
  data: JSON.stringify(jsonData),
  success: function(result) { ... },
  error: ...
});
```

### 6.19.2 Example Response

```
[
  {
    "tree_acc": "TGDR069-190112",
    "name": "flush",
    "uniquename": "flush-190112",
```

(continues on next page)

(continued from previous page)

```

    "phenotype_name": "Budset scoring",
    "value": "119.6",
    "units": "days",
    "po_name": " bud burst stage",
    "po_accession": "PO:0025532",
    "pato_name": "time",
    "pato_accession": "PATO:0000165",
    "cvterm_name": "Budset scoring",
    "cvterm_accession": "CO_357:1000010"
  }
  ...
]

```

### 6.19.3 Response Fields

Field	Type	Description
tree_acc	string	The id of the tree
name	string	The name of the phenotype
uniquename	string	Uniquename of the phenotype assigned by chado
phenotype_name	string	The phenotype name
units	string	The units of the value measured
po_name	string	The plant ontology name of the tree
pato_name	string	Phenotype properties
pato_accession	string	Accession of the pato
cvterm_name	string	Ontology term
cvterm_accession	string	Ontology term accession

## 6.20 GET TREE GENOTYPES

Get the genotype information of the tree

Type of Request: GET

URL: [http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/genotypes?api\\_key=<api\\_key>&tree\\_id=<tree\\_id>](http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/genotypes?api_key=<api_key>&tree_id=<tree_id>)

Query string parameters:

Name	Type	Description
api_key	string	Your API key
tree_id	string	The unique id of the individual tree

### 6.20.1 Example GET request using ajax

```

$.ajax({
  url: 'https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/genotypes?api_key=<api_
  ↪key>&tree_id=TGDR069-190112',
  type: 'GET',
  contentType: 'application/json',

```

(continues on next page)

(continued from previous page)

```
data: JSON.stringify(jsonData),
success: function(result){ ... },
error: ...
});
```

### 6.20.2 Example Response

```
[
  {
    "tree_acc": "TGDR001-7014"
    "uniquename": "SNP-PTRI.339.C1-162-Potr-A:A"
    "marker_name": "-"
    "description": "A:A"
    "marker_type": "SNP"
    "study_type": "Natural Population (Landscape)"
  }
  ...
]
```

### 6.20.3 Response Fields

Field	Type	Description
tree_acc	string	The id of the tree
uniquename	string	Uniquename of the phenotype assigned by chado
marker_name	string	The marker name
description	string	The description
study_type	string	The study type

## 6.21 GET TREE MARKERS

Get the marker types and additional genotype information associated with the tree

Type of Request: GET

URL: [http://tgwebdev.cam.uhc.edu/cartogratree/api/v2/markertypes?api\\_key=<api\\_key>&tree\\_id=<tree\\_id>](http://tgwebdev.cam.uhc.edu/cartogratree/api/v2/markertypes?api_key=<api_key>&tree_id=<tree_id>)

Query string parameters:

Name	Type	Description
api_key	string	Your API key
tree_id	String	The unique id of the individual tree

### 6.21.1 Example GET request using ajax

```
$.ajax({
  url: 'https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/markertypes?api_key=<api_
  ↪key>&tree_id=TGDR006-14454',
  type: 'GET',
  contentType: 'application/json',
  success: function(result){ ... },
  error: ...
});
```

### 6.21.2 Example Response

```
[
  {
    "tree_acc": "TGDR006-14454",
    "marker_name": "0_10207_01_333-Pita",
    "description": "C:C",
    "marker_type": "SNP",
    "study_type": "GxP",
    "quality_score": null,
    "marker_technology": "array",
    "assembly_type": null,
    "scaffold": null,
    "position": null,
    "original_name": null
  }
  ...
]
```

### 6.21.3 Response Fields

Field	Type	Description
tree_acc	string	The id of the tree
marker_name	string	The name of the marker
description	string	The description of the genotype
marker_type	string	The type of the marker
study_type	string	The type of the study
quality_score	unknown	The quality score
marker_technology	string	The technology of the marker
assembly_type	unknown	The assembly type
scaffold	unknown	The scaffold of the genotype
position	unknown	The position of the genotype
original name	unknown	The original name of the genotype

## 6.22 GET TREE PUBLICATIONS

Get the publication information for a tree if it has a publication associated with it

Type of Request: GET

URL: `http://tgwebdev.cam.uhc.edu/cartogratree/api/v2/publications?api_key=<api_key>&tree_acc=<tree_acc>`

Query string parameters:

Name	Type	Description
api_key	string	Your API key
tree_acc	String	Study accession of a tree

### 6.22.1 Example GET request using ajax

```
$.ajax({
  url: 'https://tgwebdev.cam.uhc.edu/cartogratree/api/v2/publications?api_key=<api_
  ↪key>&tree_acc=TGDR002',
  type: 'GET',
  contentType: 'application/json',
  success: function(result){ ... },
  error: ...
});
```

### 6.22.2 Example Response

```
[
  {
    "project_id": "45",
    "accession": "TGDR002",
    "pub_id": "21371",
    "title": "Astonishingly low genetic variation in Quercus acutissima, an
    ↪important tree species in Satoyama, a traditional Japanese rural forest and
    ↪agricultural landscape, revealed by chloroplast microsatellite markers",
    "author": "Tsuda, Yoshiaki",
    "species": "Quercus acutissima",
    "tree_count": "2152",
    "phen_count": null,
    "ontology_ids": null,
    "phenotypes_assessed": null,
    "gen_count": "12912",
    "markers": "cpSSR",
    "study_type": "G",
    "volumetitle": null,
    "volume": "",
    "series_name": "Tree Genetics & Genomes",
    "issue": "", "pyear": "2012",
    "pages": "-",
    "miniref": null,
    "uniquename": "Saito, Yoko, Tsuda, Yoshiaki , Uchiyama, Kentaro , Fukuda,
    ↪Tomohide , Seto, Yasuhiro , Kim, Pang , Ide, Yuji Astonishingly low genetic
    ↪variation in Quercus acutissima, an important tree species in Satoyama, a
    ↪traditional Japanese rural forest and agricultural landscape, revealed by
    ↪chloroplast microsatellite markers 2012; () -",
    "type_id": "229",
    "is_obsolete": false,
    "publisher": null,
    "pubplace": null
  }
]
```

(continues on next page)

(continued from previous page)

```
}
]
```

### 6.22.3 Response Fields

Field	Type	Description
project_id	integer	The id of the project
accession	string	The accession of the publication
pub_id	integer	The id of the publication
title	string	The title of this study
author	string	The primary author fo this study
species	string	The species focused on by this study
tree_count	integer	Number of trees assessed by this study
phen_count	intger	Number of phenotypes for this study
ontology_ids	string	The ids of the associated ontologies
phenotypes_assessed	integer	The phenotypes assessed
gen_count	integer	Number of genotypes for this study
markers	string	Markers recorded for the species of this study
study_type	string	The type of the study
volumetitle	string	Title of volume where this study was published
volume	string	The volume of the paper published
series_name	string	The series of the published paper
issue	string	The issue of the published paper
pages	string	The pages of the published paper
miniref	string	The mini reference to the paper
uniquename	string	The uniquename of the paper
type_id	integer	The type_id of the paper
is_obselete	boolean	Is the paper obselete
publisher	string	The publisher of the paper
pubplace	string	The place???

## 6.23 GET TREE TREESNAP COMPLETE

TreeSnap trees are accessed through the TreeSnap API provided by <https://treesnap.org> Please visit the site, download their app and submit your own trees if you want to contribute both to TreeSnap and CartograTree

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/treesnap?api_key=<api_key>&tree_id=<tree_id>`

Query string parameters:

Name	Type	Description
api_key	string	Your API key
tree_id	string	The unique id of a treesnap tree

### 6.23.1 Example GET request using ajax

```
$.ajax({
  url: 'https://tgwebdev.cam.uchc.edu/cartogratree/api/v2/treesnap?api_key=<api_key>
  ↳&tree_id=treesnap.1',
  type: 'GET',
  contentType: 'application/json',
  success: function(result){ ... },
  error: ...
});
```

The response will be the same as accessing the TreeSnap API directly for individual trees. To see the full documentation for TreeSnap’s API please go to: <https://github.com/statonlab/Treesnap-website/wiki/Public-API-Documentation>

## 6.24 GET SESSION DATA

Gets saved session state properties

Type of Request: GET

URL: `http://tgwebdev.cam.uchc.edu/cartogratree/api/v2/user/session?api_key=<api_key>&session_id=<session_id>`

Query string parameters

Name	Type	Description
api_key	string	Your API key
session_id	string	The id of the session

### 6.24.1 Example Response

```
{
  "layers":null,
  "filters":null,
  "zoom":null,
  "pitch":null,
  "bearing":null,
  "center":null,
  "excluded_trees":null,
  "session_id":<session_id>
}
```



## 6.24.2 Response Fields

Field	Type	Description
api_key	string	Your API key
session_id	string	The session id
layers	object	The layers activated in this session
filters	object	The filters and active datasets
excluded_trees	array	Trees excluded for analysis
zoom	double	Map zoom property
pitch	double	Map pitch property
bearing	double	Map bearing property
center	array of doubles	Center of map [long, lat]

## 6.25 GET USER SESSION

Gets session information for a saved session by the user

Type of Request: GET

URL: `http://tgwebdev.cam.uhc.edu/cartogratree/api/v2/user/session/by-user?api_key=<api_key>&user_id=<user_id>&session_id=<session_id>`

Query string parameters

Name	Type	Description
api_key	string	Your API key
user_id	integer	The id of user created by drupal
session_id	string	The id of the session of the user

### 6.25.1 Example Response

```
[
  {
    "user_id": "186",
    "title": "ok",
    "comments": "",
    "session_id": "0b6d71c1324b3bfc68732caaa9315c27"
  }
]
```

### 6.25.2 Response Fields

Field	Type	Description
title	string	The title of the session
comments	string	The comments by the user
user_id	integer	The id of the user
session_id	string	The session id

## 6.26 GET ALL USER SESSIONS

Gets all the saved sessions by a user along with the saved session state properties

Type of Request: GET

URL: `http://tgwebdev.cam.uhc.edu/cartogratree/api/v2/user/sessions/by-user/all?api_key=<api_key>&user_id=<user_id>`

Query string parameters

Name	Type	Description
api_key	string	Your API key
user_id	integer	The id of user created by drupal

### 6.26.1 Example Response

```
[
  {
    "title": "t2",
    "comments": "",
    "session_id": <session_id>,
    "layers": {},
    "filters": {
      "query": {
        "condition": "OR",
        "rules": [
          {
            "id": "genus",
            "field": "genus",
            "type": "string",
            "input": "select",
            "operator": "equal",
            "value": "Abies"
          }
        ]
      },
      "active_sources": ["0", "1", "2"]
    },
    "zoom": 4,
    "pitch": 0,
    "bearing": 0,
    "center": {"lng": 24.851, "lat": 41.922},
    "created_at": "2019-07-01T19:08:32.010Z", "excluded_trees": {}
  }
  ...
]
```

## 6.26.2 Response Fields

Field	Type	Description
title	string	The title of the session
comments	string	The comments of the session
session_id	string	The session id
layers	object	The layers activated in this session
filters	object	The filters and active datasets
zoom	double	Map zoom property
pitch	double	Map pitch property
bearing	double	Map bearing property
center	array of doubles	Center of map [long, lat]
created_at	datetime	Date the session was saved