NORTHWEST NAZARENE UIVERSITY

Training Data Selector

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Gregory Smith
2018

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Gregory Smith
2018

Training Data Selector

Author: [illegible]
Gregory Smith

Approved: [illegible]
Dale Hamilton, Ph.D., Assistant Professor of Computer Science,
Department of Mathematics and Computer Science, Faculty Advisor

Approved: Steve Shaw
Steve Shaw, Ph.D., Professor of Political Science, Department of History and Political Science, Second Reader

Approved: Barry Myers
Barry L. Myers, Ph.D., Chair,
Department of Mathematics & Computer Science

ABSTRACT

Creating a Simple and Easily Accessible Training Data Tool for the FireMAP Supervised Classifiers.

SMITH, GREGORY (Department of Mathematics and Computer Science), MYERS, DR. BARRY (Department of Mathematics and Computer Science), HAMILTON, DR. DALE (Department of Mathematics and Computer Science).

Gathering training data for a pixel based machine learning classifier can be a painstaking slow and tedious task. Not only must the user ensure the data being gathered is accurate, but they must also gather enough data to successfully train the classifier. The Training Data Selector (TDS) allows a user to accurately and quickly produce training data. This tool provides accurate training data for analytics as diverse as wildland fire management and pathology. The TDS allows the user to draw on data in any web browser, label that data, and then extract and export the pixel data, thus allowing a classifer to learn what that picture is as well as images like it. This application utilizes human expertise without compromising computer processing power.

As well as providing a quick and clean solution to extracting information from data to be used in various supervised classifiers, the TDS application was built and designed for users who are inexperienced with computer applications and therefore provides a simple, easy, and intuitive interface for all users on all platforms. The TDS provides the greatest flexibility, power, and availability to extract the selected data the user chooses for training a supervised classifier.

Acknowledgements

To begin with I would like to thank my wonderful fiancé Calli for listening to all the software problems I encountered, allowing me to think out loud and scribble down ideas in the middle of our dates. Without her the Training Data Selector would not be where it is at today. I would also like to thank Gregory Athons from Curious Media. Without his help the drawing and zoom tool would not be as efficient as they are currently. Next, I want to thank David Harris and Isaac Kronz for their help in getting the FireMAP portal up on the Computer Science servers, while getting it live is still a work in progress I know without them this project would still be running on the local machine long after I graduated. Next to last, I would like to thank the Institutional Development Award (IDeA) from the National Institute of General Medical Sciences of the National Institutes of Health which helped fund this project under Grant #P20GM103408 and the NASA Idaho Space Grant Consortium (ISGC) which also helped fund this project under Grant # FPK900-SB-040. Finally, I would like to, of course, thank my professors, Dr. Myers and Dr. Hamilton. Without their guidance and input before, during, and after development, the Training Data Selector would not even exist.

Table of Contents

Title Page i
THESIS ii
Signature Page ii
ABSTRACT iii
Acknowledgements iv
Table of Contents v
Overview 1
Background 2
Exploration 4
Design 5
Implementation 11
Future Work 17
Conclusion 18
References 20
Appendix A 22
- *server.js 22*
- *auth 23*
  - hasher.js 23
  - index.js 23
- *controllers 27*
  - upload_imagery_controller.js 27
  - download_imagery_controller.js 27
  - home_controller.js 28
  - index.js 28
  - logout_controller.js 28
  - reports_controller.js 29
  - training_data_selector_controller.js 29
- *FireMAPCoordinateSelector 32*
- *data 60*
  - database.js 60
  - index.js 61
- *public 62*
  - css 62
    - create_account.css 62
    - site.css 65
  - js 76
    - login.js 76
    - training_data_selector.js 78
    - create_account.js 205
    - layout.js 208
- *views 209*
  - create_account_page.vash 209
  - download_imagery.vash 211
  - index.vash 211
  - layout.vash 212
  - login_page.vash 213
  - reports.vash 214
  - training_data_selector.vash 215
  - upload_imagery.vash 223
- *README.md 224*

List of Figures

Figure 01 – Mockup of the layout for the Training Data Selector **5**
Figure 02 – The point tool selecting individual burn pixels **6**
Figure 03 – The pencil tool selecting burn pixels **6**
Figure 04 – The line tool selecting burn pixels **6**
Figure 05 – The polyline tool selecting burn pixels **7**
Figure 06 – The polygon tool selecting burn pixels **7**
Figure 07 – The circle tool selecting burn pixels **7**
Figure 08 – Functionality buttons **8**
Figure 09 – The FireMAP database ERD before changes **9**
Figure 10 – The new FireMAP contact and organization tables **10**
Figure 11 – The Training Data Selector before uploading an image **13**
Figure 12 – The Training Data Selector after uploading an orthomosaic image **13**
Figure 13 – GUI for the Coordinate Selector's command line arguements **15**

Overview

The primary goal of this project was to create an application that would enable users to quickly extract and label training data from remotely sensed imagery. The secondary goal was to develop a FireMAP portal website that the Training Data Selector would be hosted from as well as other FireMAP tools. The training application would need to allow users to draw over an image, label those drawings, and then extract the pixel locations from their drawings and attach the label associated with that drawing to each pixel coordinate. This labeled pixel list would need to be in a format that the already written supervised classifiers could utilize.

Background

FireMAP or Fire Monitoring and Assessment Platform is a tool that is designed to make the jobs of post-fire assessment teams easier and safer. After a wildland fire has been extinguished, it is essential to assess how much damage it caused to the ecosystem. Burned Area Emergency Response (BAER) teams must go out and record the extent of the fire as well as determine the fire severity. If the fire were a high-intensity fire, meaning there was high biomass consumption, it would take longer for the land to recover. Therefore, in a high intensity burned area, the BAER team may implement various methods to stabilize and rehabilitate the area, mitigating the adverse effects of a high severity burn. These methods include but are not limited to planting new grass and trees, installing water or erosion control devices, replace burned wildlife habitat, and treat pre-existing noxious weed infestations (USDA Forest Service, 1999.

Current methods BAER teams use to gather the information they need to make these decisions is dangerous and slow – requiring individuals to walk acres upon acres of burned land where a destabilized tree may fall at any time. FireMAP makes their jobs easier and safer via the use of machine learning and sUAS By using sUAS to fly the recently burned land, the BAER team could take pictures of the area and then use the FireMAP algorithms to determine the extent and severity of the burn, thus reducing the amount of time the BAER team spends accessing the land and minimizing their risk.

The algorithms FireMAP use must be told what a pixel is before it can classify it. For example, if a pixel in an image is burned or unburned and if it is burned, is it white ash representing high severity burn or black ash representing a low severity burn. The need for these algorithms to be told what something is first is where the need for a good training set comes in. If the classifiers are fed lousy training data, then they will output bad results (Hamilton, 2017b). The Training Data Selector has the flexibility to tell a classifier what any pixel is. This means it can help a classifier distinguish between burned and unburned, canopy, surface fuel, and bare earth (Hamilton, 2018a)(Hamilton, 2017a), and even help in archaeoinformatics (Hamilton, 2018b).

Previously, training data had to be manually created by opening a photo editor application such as GIMP extracting a rectangular region from the image, and then manually labelling this region. While this method worked, it was highly inefficient as only a rectangle could be used to select training data, and the process was slow and tedious. Therefore, the new training data selector method had to be easy to use, quick, and accurate. The Training Data Selector not only meets these requirements, but it also provides ease of access by being an online tool (Hamilton, 2018a).

The FireMAP portal, firemap.cs.nnu.edu, which is where the Training Data Selector is hosted at, needed to provide a place for individuals outside of Northwest Nazarene University (NNU) to interact with the FireMAP and its tools.

Exploration

Before the Training Data Selector could be developed, research had to be done on the best way to implement a program that could draw, label, and extract pixels from an image. It ultimately came down to two languages, JavaScript or C#. Both had tools that allowed for the smooth implementation and interaction of a graphical user interface (GUI), and both could to upload an image and see individual pixels. C# had the benefit of having a faster response time to user input, but JavaScript had the advantage of being web-based. Ultimately, it was decided that the Training Data Selector would be created with JavaScript so it could be placed on the NNU servers, thus eliminating the need for users to download a program onto their machine.

After deciding to use JavaScript, further exploration began on what JavaScript libraries could be utilized to help speed up development. It soon became apparent that Node.js would be best for server-side implementation due to its ability to quickly interface with client-side JavaScript. It was also discovered that CreateJS was the perfect match for drawing on images as it already had pre-written functions for drawing and discovering pixel locations.

Design

The design and implementation part of this project are not easily separable as the design changed based upon user feedback from completed implementation pieces. However, some things that did not change. For example, it was important for the user to be able to select what tool to draw with, have a drawing region, and can pick what color the drawing tool should be. As using a black pen on black ash did not stand out very well. With these constraints, the application was split into three pieces. The tool window, the image window, and the color window.

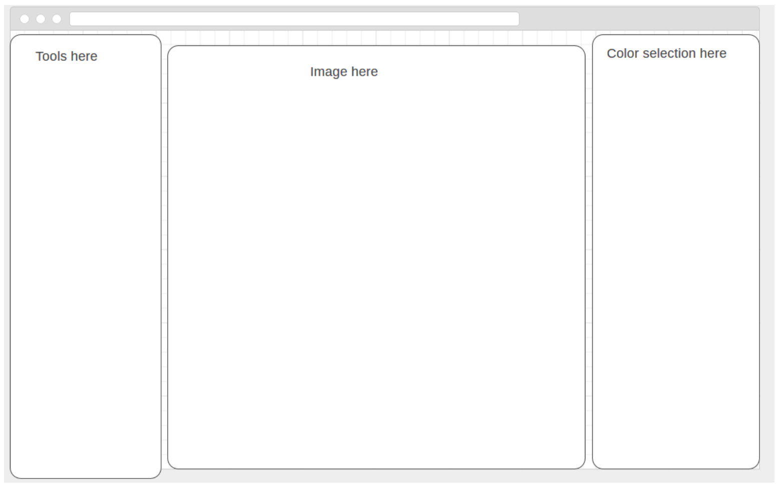


Figure 01 – Mockup of the layout for the Training Data Selector

There would be eight tools the user could select to draw with. Each tool having different strengths and weaknesses. These eight tools and their desciptions can be found below.

- Point Tool:
    - Allows the user to select a single pixel to be labeled.
    - Is highly accurate but impractical for selecting large groups of pixels.

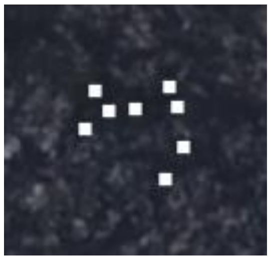


Figure 02 – The point tool selecting individual burn pixels

- Pencil Tool:
    - Allows the user to freely select the pixels to be labeled.
    - Is highly flexible but hard to control.

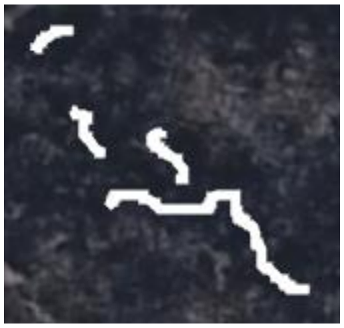


Figure 03 – The pencil tool selecting burn pixels

- Line Tool:
    - Allows the user to select a line of pixels to be labeled
    - Less flexible than the pencil tool, but gives the user more accuracy.

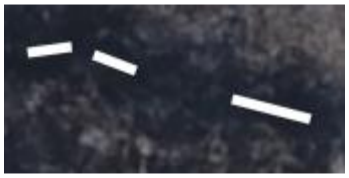


Figure 04 – The line tool selecting burn pixels

- Polyline Tool:
    - Allows the user to select multiple connected lines of pixels to be labeled.
    - Grants the flexibility of the pencil tool with the accuracy of the line tool.

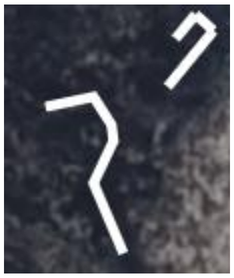


Figure 05 – The polyline tool selecting burn pixels

- Polygon Tool:
    - Allows the user to select multiple connected lines of pixels and the area within these lines to be labeled.
    - Grants the same utility of the polyline tool with the added bonus of grabbing the pixels inside the created polygon.

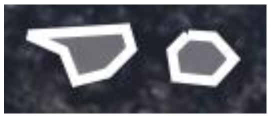


Figure 06 – The polygon tool selecting burn pixels

- Circle Tool:
    - Allows the user to select a circle of pixels and the pixels inside the circle to be labeled.
    - Grants the same utility as the polygon tool, but can only create a circle instead of any shape.

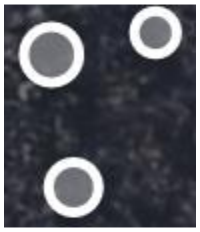


Figure 07 – The circle tool selecting burn pixels

- Auto-Cluster Tool:
    - When implemented this tool will use clustering techniques to determine training regions on the image.
    - Once these training regions are generated, it would be the user's job to go through and label each region.

- Flood-Fill Tool:
    - When implemented this tool will allow the user to select a pixel like the point tool, but it will also grab all similar pixels in the region, expanding its pixel acceptance with each click.
    - This tool will allow for quick and easy selection of pixels based upon their spectral reflectance, and then the user would only have to label these similar pixels instead of finding them too.

After noticing that my user created an error in drawing and could not remove that error without refreshing the page, it was determined that a Save, Undo, Redo, Delete, and Load buttons and their functionality would need to be implemented. The Save button is what allows the user to save their drawing data and receive the .csv file containing their labels and the pixels attached to those labels. The Undo button undoes the most recently drawn shape. The Redo button redraws the most recently undrawn shape. The Delete button will delete any drawing selected with the mouse. Finally, the Load button allows a user to load previously drawn data onto the image it was originally created with to add or remove drawings to the data.

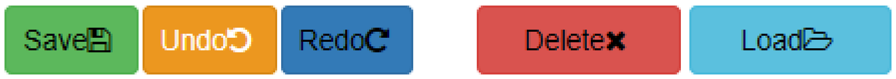


Figure 08 – Functionality buttons

The next to last part of the design phase was updating the FireMAP database which was previously worked on by Peter Oxley as a semester project for the COMP3330 Database Design and Programming classes here at NNU (personal communication, April 6, 2017) . The amount of information we were collecting on users was excessive and could be reduced. Below you can find the original schema for the database.

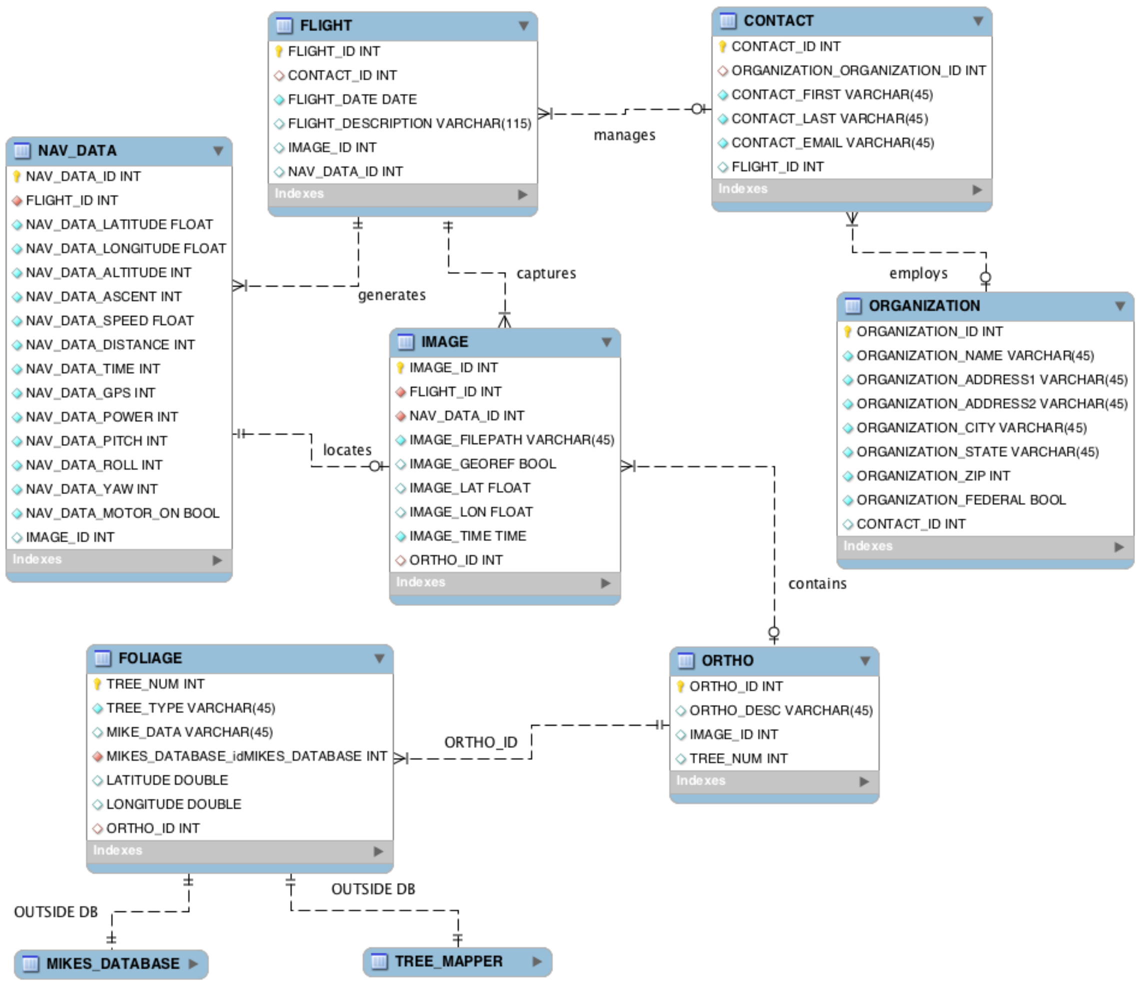


Figure 09 – The FireMAP database ERD before changes

The final part of the design phase was organizing what information or what web pages would be on the FireMAP portal for users to access. It was determined that they should only be able to see the about page, the login, and the create account page without first being logged in. Then once they were logged in, they could access the Training Data Selector, upload imagery, download imagery, or generate reports. It was discovered during the design phase that FireMAP was collecting more information from users than it needed to, so the contact table and organization table got updated to the following.

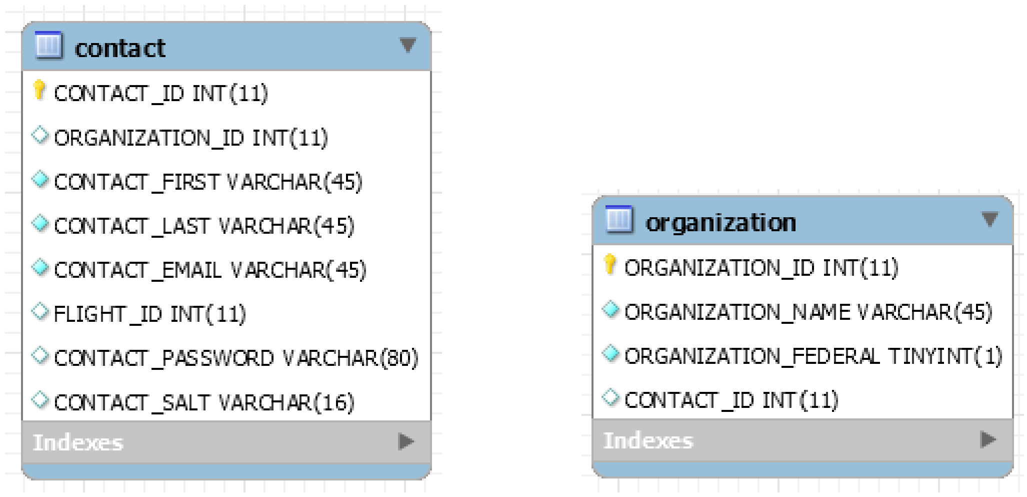


Figure 10 – The new FireMAP contact and organization tables

Implementation

Being the first goal of this project, the Training Data Selector was implemented first. It was crafted with CreateJS, which is a JavaScript library that allows for drawing on a canvas. This canvas can be a blank window, or it can be an image. So, after setting up the generic layout of the Training Data Selector, the upload image functionality was implemented – turning the image pushed into the browser into a CreateJS canvas that could be drawn on.

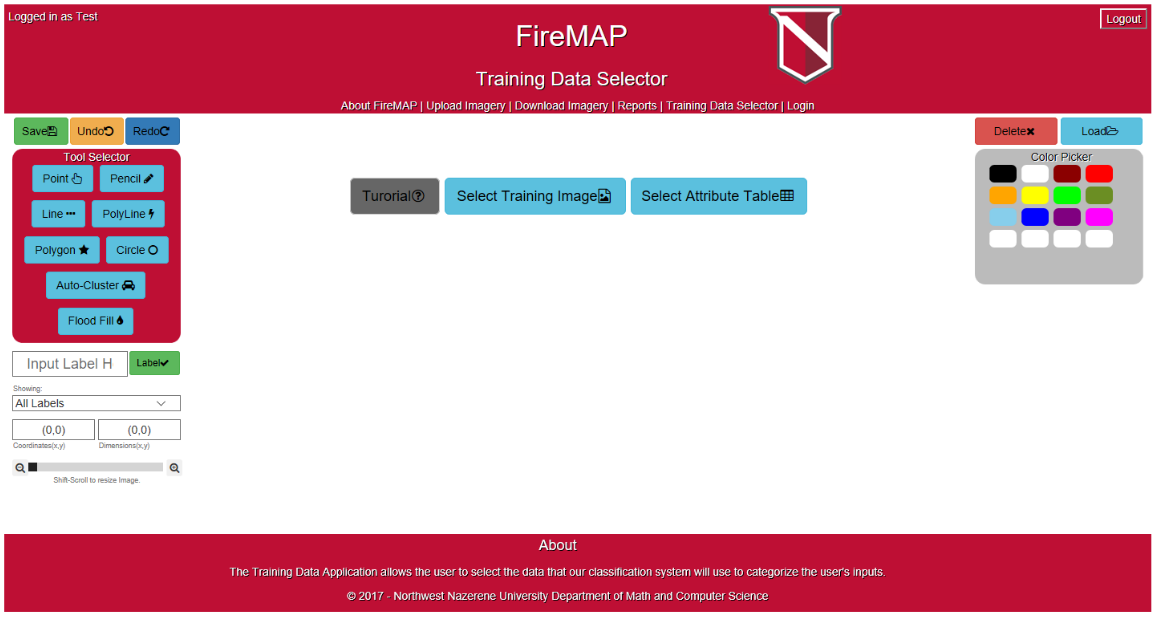
From there, the drawing tools were implemented. These tools were created, just like the canvas, with the help of a CreateJS. Each tool's base method of working is essentially the same – when the mouse is clicked, place a point at that location. With the point tool, the drawing ends after this. However, with all other currently implemented drawing tools, a line is then drawn from the first point to the location of the next point – which is determined by another mouse click. The area tools, such as the polygon and the circle are based upon the same principle, but also make sure shade in the area inside the lines to communicate to the user that they have selected an area and not just a line.

After the tools were implemented, changing the color of the drawings was the next problem to conquer. This functionality, once again, was done with the help of CreateJS as it had a feature for changing the pen color. However, to make the change available to the user, color buttons, were implemented, as well as a color wheel in case the colors provided were not a wide enough range of colors.

The next stage of the Training Data Selector development was labeling the drawings the user created. A text box was built, allowing the user to enter their label and then label every unlabeled drawing on the canvas at once. Implementing the labeling system this way allowed the user to label multiple drawings at once. If a drawing was accidentally labeled the wrong thing, it could just be deleted and redrawn. Once again, while showing the primary user what had been created so far, it was brought to attention that the user had no way of knowing what a drawing was labeled after they labeled it. Thus, the design phase was revisited and a solution brought forward and implemented. Since every drawing is a JavaScript object, it was possible to call information on that object after it was created. When a drawing is clicked on by a user, that information is then called up, showing the user what the drawing is labeled. This way of crafting drawings turned out to be especially helpful when a user loaded in previously drawn data as they can then still see what they labeled days ago.

With drawing implemented it was now time to apply the most critical part of this project – extracting the drawing pixel locations and their label from the image. The point tool was the easiest to obtain information from as those coordinates were already known. The line based tools were a bit more challenging but still simple as the endpoints were known and the rest could be calculated using the slope of the line. The biggest hurdle here was determining the area of a potentially irregular polygon. After some research, it was decided that a method called ray-casting would be the best option (Mecki, 2008).

Ray casting shoots a horizontal ray across the canvas, tallying every time it intersects with a line. If the ray intersects an odd number of times, then it is within the polygon, if it crosses an even number of times, then it is outside the polygon. Using this method, it was possible to determine every pixel that was inside a user drawn polygon and attach the correct label to it. However, doing this in JavaScript took minutes and often crash the current web browser. It was therefore determined that JavaScript would just calculate the vertices of every drawn shape and attach a label to that shape. The Coordinate Selector, which was implemented in C++, took these vertices and labels and outputted every pixel and their label into a .csv file which current FireMAP classifiers could utilize. The finished Training Data Selector can be shown below.



The Training Data Application allows the user to select the data that our classification system will use to categorize the user's inputs.

© 2017 - Northwest Nazerene University Department of Math and Computer Science

Figure 11 – The Training Data Selector before uploading an image

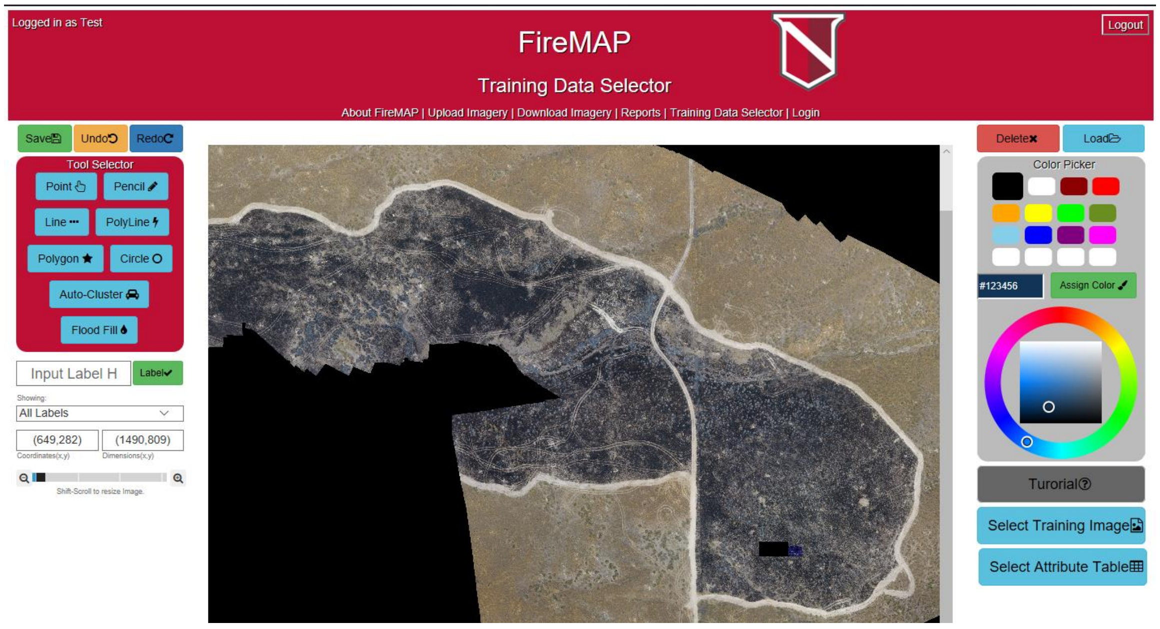


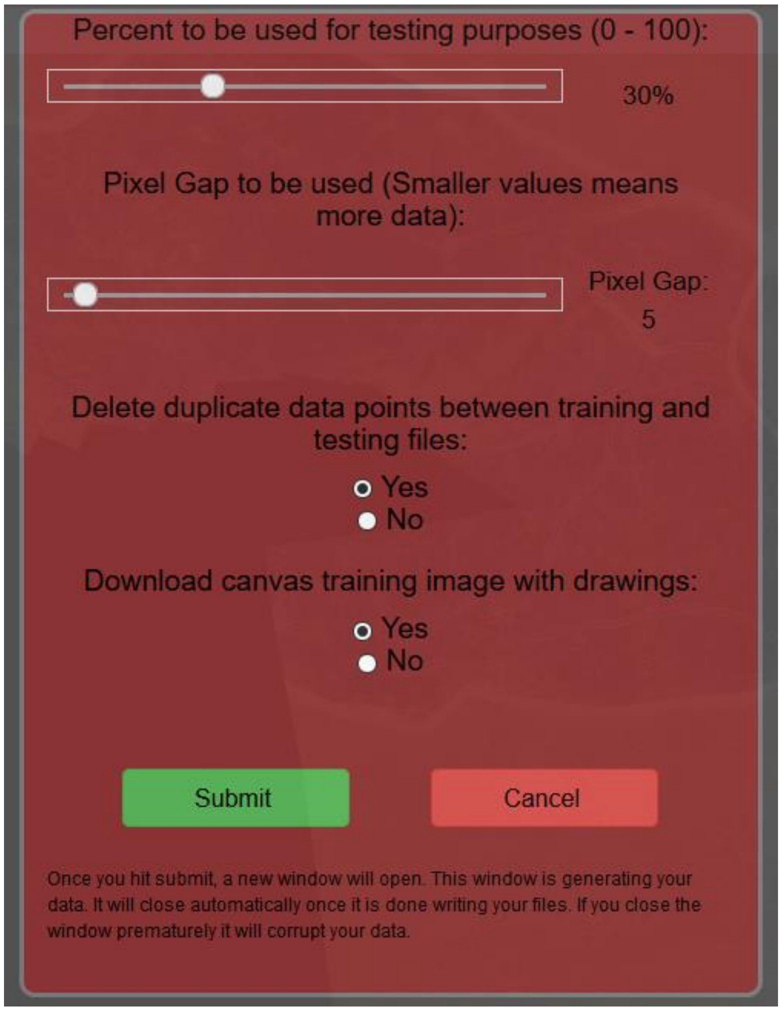
Figure 12 – The Training Data Selector after uploading an orthomosaic image

With the training data portion of the project done, it was time to start part two of this project – the FireMAP portal website. Using Node.js – a backend JavaScript library – the Training Data Selector was combined with previous web pages made in Web Development class at NNU in Fall 2015 and became the FireMAP portal. Since these web pages were created by different groups and their functionality was not fully implemented, they could not be directly ported into the portal. All pages, the login and create account page, the download imagery, and the upload imagery page needed to be changed visually. The download and upload pages' functionality had to be scrapped as FireMAP does not have the database capabilities as of yet to handle image storage. The login and create account page had to be rewritten as it was first written in PHP.

A scripting language called Vash was used to make all the pages match. Vash works with Express which works with Node.js, to create HTML that can be similar across all pages and to allow variables to passed into HTML. Using Vash allowed the same generic layout to be used across multiple pages while providing the flexibility to customize pages as needed.

The next step was the account creation and login pages. The PHP code was stripped out, and Node.js code was implemented. Node packages like mysql were pulled in to allow a connection to the FireMAP mysql database and to select and pass information to said database. When it came to account creation, passwords were salted and then hashed with SHA-256. Salting the password adds random characters to the user's password, making it more difficult to crack and encypting with SHA-256 meant the password would not be sent in plaintext across the internet. Both are essential for information security and both were implemented with Passport, an authentication middleware for Node.js (Passport, 2018). The login password would then be salted and hashed as well. If the password hash matched the database password hash, the user was then logged in. Logging in was an essential part because a user cannot access the Training Data Selector, Upload Imagery, Download Imagery, and Reports page without first logging in.

Node.js also allowed the transition from the Training Data Selector to the Coordinate Selector to happen behind the scenes. Once the user hits the Save button, they are prompted to enter the command line arguments for the Coordinate Selector via the graphical user interface (GUI) seen below.



Percent to be used for testing purposes (0 - 100):

30%

Pixel Gap to be used (Smaller values means more data):

Pixel Gap: 5

Delete duplicate data points between training and testing files:

[✓ Yes] No

Download canvas training image with drawings:

[✓ Yes] No

Once you hit submit, a new window will open. This window is generating your data. It will close automatically once it is done writing your files. If you close the window prematurely it will corrupt your data.

Figure 13 – GUI for the Coordinate Selector's command line arguements

Then, once they hit submit on the popup interface, the vertices generated by the Training Data Selector along with the additional inputs just specified by the user are sent to the Coordinate Selector program. Next, Node.js waits asynchronously for the Coordinate Selector to finish writing its output files. The Coordindate Selector generates two .csv files. Both follow the format of giving the user label, x pixel coordinate, and y pixel coordinate (e.g. Burned, 5, 15). The first file is meant for training the classifiers. The second file is meant for validating the classifiers. The percentage of data in the training file verses the validatation file depends on the user arguments given earilier in the graphical interface. The validation file is how the user can determine whether or not their classifier is outputting accurate results. As the user knows what this data should be labeled they can check the output of the classifier to determine if the classifier agrees with them. Once the files are generated, another Node package, archiver, is used to zip the .csv files and send them as a download option to the user.

Future Work

There are a few things that still need to be done before this project can be completed. The first, and most critical is placing this project on the NNU Computer Science servers so that anyone can access it. This is currently being worked on and should be done by the end of the year.

Other improvements involve implementing the Flood-Fill tool and the Auto-Cluster tool. With those two tools completed, the Training Data Selector will be fully completed.

As for the portal, the Reports page, which would be used to generate reports created by the FireMAP classifiers, still needs to be implemented. The upload and download imagery pages need to be implemented as well.

To finish the portal, more work will need to be done on the database to incorporate images and who can access what images. For example, should a user be able to access all FireMAP photos, or only the ones they created? Should FireMAP host these images or should they be outsourced to a cloud server that specializes in image storage? These questions and others like it must be answered before further work in this area can be continued on the FireMAP portal.

Conclusion

When this project first started FireMAP had a problem that it needed to solve. They needed a way to create training data fast and accurately. The Training Data Selector fulfilled this goal as no one has gone back to the old system of selecting pixels in other tools such as GIMP (Dale Hamilton, personal corspondance, April 13, 2018) . As a whole, the project was a success. Even if the FireMAP portal does not get put on to the Computer Science servers here at NNU, users can still download the Training Data Selector and use it locally on their machine. Installation instructions can be found in the README.md file inside the downloaded zip folder that contains the Training Data Selector and in Appendix A. Considering when I started that I knew very little about JavaScript and nothing about Node.js, this is a significant achievement. I learned that even though JavaScript is not an object-oriented language, it is still possible to mimic object-oriented code in JavaScript by creating essentially a multivariable array, which is what my drawing objects are. I also learned how Node.js could be used to run a C++ program and return that data to the user via a download, examples of doing so can be found in the training_data_selector_controller.js file in Appendix A. Node.js can be used to connect to a database, query it, salt and hash passwords. Example code for escaping user input and querying a database in Node.js can be found in Appendix A, database.js and index.js in the authentication (auth) folder. The index.js file also contains the code for salting and hashing user passwords. More example code can be found at the official documentation page for Node and Node packages which can be found on the references page.

There were many times during this project that I struggled. I struggled with learning new libraries and what packages needed to be installed and asynchronous callback functions. I struggled with debugging my code using alerts and console statements instead of breakpoints. In the end, these struggles made me a better programmer. They taught me to seek help, to ask questions, and most importantly, not to give up, because eventually, even the hardest problems get solved.

References

Ctalkington. (2018). Nodejs archiver Doc. Retrieved March 22, 2018, from https://www.npmjs.com/package/archiver

Express. (2018). Express Doc. Retrieved March 22, 2018, from http://expressjs.com/en/api.html

Mecki. (2008, October 20). How can I determine whether a 2D Point is within a Polygon? Retrieved April 10, 2017, from https://stackoverflow.com/questions/217578/how-can-i-determine-whether-a-2d-point-is-within-a-polygon

Node.js. (2018). Nodejs Docs. Retrieved March 22, 2018, from https://nodejs.org/en/docs/

Nodejs mysql. (2018). Nodejs mysql Doc. Retrieved March 22, 2018, from https://github.com/mysqljs/mysql

Passport. (2018). Documentation. Retrieved April 10, 2018, from http://www.passportjs.org/docs/

USDA Forest Service. (1999, January 1). Burned Area Emergency Response, BAER. Retrieved March 22, 2018, from https://www.fs.fed.us/biology/watershed/burnareas/background.html

W3Schools. (n.d.). W3Schools. Retrieved March 22, 2018, from https://www.w3schools.com/

Hamilton, D; Ph.D., University of Idaho, Moscow, ID, Computer Science, May 2018 *Improving Mapping Accuracy of Wildland Fire Effects from Hyperspatial Imagery using Machine Learning.* (Doctoral dissertation).

Hamilton, D; Myers, B; Hamilton, N*, (in press) "Evaluation of Image Spatial Resolution for Machine Learning Mapping of Wildland Fire Effects", *IEEE Intelligent Systems Conference 2018;* September 6-7, 2018; London, UK.

Hamilton, D; Myers, B; Branham, J*, (2017) "Evaluation of Texture as an Input of Spatial Context for Machine Learning Mapping of Wildland Fire Effects". *Signal and Image Processing: An International Journal,* 8(5).

Hamilton,D; Bowerman, M*; Colwell, J; Donahoe, G; Myers B, (2017) "A Spectroscopic Analysis for Mapping Wildland Fire Effects from Remotely Sensed Imagery", *Journal of Unmanned Vehicle Systems,* Virtual Issue, DOI:10.1139/juvs-2016-0019.

Hamilton, D; Pacheco, R*, (2018) Archaeoinformatics: Computational Archeology*: A Lot of Tech to Find an Old Tin Can*, NW Anthropological Conference, March 30, 2018, Boise, ID (presentation)

Appendix A

server.js

```
'use strict';
//var http = require('http');
var database = require('./data/index.js');
const port = process.env.PORT || 1337; //Use Port 1337 (What
process.env.PORT is on my machine) if process.env.PORT is not set
const http = require('http');
const myLogMod = require('./server_modules/log.js');
const express = require('express');
const bodyParser = require('body-parser');
const flash = require('connect-flash');
const session = require("express-session");
const app = express();
var controllers = require('./controllers/index.js');

app.set("views", __dirname + "/views");
app.set("view engine", "vash");
app.use(express.static(__dirname + "/public"));
app.use(bodyParser.urlencoded
(
    {
        extended: false,
        limit: '16mb'
    }
));
app.set('trust proxy', 1);
app.use(session({ secret: "FireMAPisthebest", resave: false,
saveUninitialized: true, cookie: { secure: false } }));
app.use(flash());

// use authentication
var auth = require("./auth/index.js");
auth.init(app);


//Map the routes
controllers.init(app);

//Database
database.init(app);


http.createServer(app).listen(port);
```

auth

hasher.js

```
//hasher.js
(function (hasher)
{
    var crypto = require('crypto');

    hasher.createSalt = function ()
    {
        var length = 8;
        return crypto.randomBytes(Math.ceil(length /
2)).toString('hex').substring(0, length);
    };

    hasher.computeHash = function (source, salt)
    {
        var hmac = crypto.createHmac("sha1", salt);
        var hash = hmac.update(source);
        return hash.digest("hex");
    };




})(module.exports);
```

index.js

```
// auth/index.js
(function(auth)
{
    var data = require("../data/database");
    var mysql = require('mysql');
    var hasher = require('./hasher');
    var passport = require("passport");
    var localStrategy = require("passport-local").Strategy;

    function userVerify(username, password, next)
    {
        password = mysql.escape(password);
        data.getUser(username, function (err, user)
        {
            if (!err && user)
            {
                var testHash = hasher.computeHash(password,
user[0].CONTACT_SALT);
                if (testHash === user[0].CONTACT_PASSWORD)
                {
                    next(null, user);
                    return;
                }
            }
            next(null, false, { message: "Invalid Credentials. Try
again."});
```

```
        });
    }

    auth.ensureAuthenticated = function (req, res, next)
    {
        if (req.isAuthenticated())
        {
            next();
        }
        else
        {
            res.redirect("/login");
        }
    }


    auth.init = function (app) {

        //setup passport authentication
        passport.use(new localStrategy(userVerify));
        passport.serializeUser(function (user, next)
        {
            next(null, user[0].CONTACT_EMAIL);
        });
        passport.deserializeUser(function (key, next)
        {
            data.getUser(key, function (err, user)
            {
                if (err)
                {
                    next(null, false, { message: "Failed to retrieve
user" });
                }
                else
                {
                    next(null, user);
                }
            });
        })
        app.use(passport.initialize());
        app.use(passport.session());

        app.get('/login', function (req, res) //Login Page
        {
            var userName = "";
            if (req.user) {
                userName = req.user[0].CONTACT_FIRST;
            }
            var passedVariable1 = "";
            var passedVariable2 = "";
            if (req.query.valid == "Successfully logged out.") {
                passedVariable2 = req.query.valid;
            }
            else {
                passedVariable1 = req.query.valid;
            }
            res.render('login_page', { title: "Login", pageHeader:
"FireMAP Portal Login Page", invalidCredsText: passedVariable1,
logoutSuccessText: passedVariable2, footerAboutText: "This is the Login
Page for the FireMAP Portal.", user: userName });

        });
        app.post("/login", function (req, res, next)
        {
            var authFunction = passport.authenticate("local", function
(err, user, info)
            {
                if (err || !user)
                {
                    if (err) { next(err); }
                    var passString = encodeURIComponent(info.message);
                    res.redirect("/login?valid=" + passString);
                }
                else
                {
                    req.logIn(user, function (err)
                    {
                        if (err)
                        {
                            next(err);
                        }
                        else
                        {
                            res.redirect("/training_data_selector");
                        }
                    });
                }
            });
            authFunction(req, res, next);

        });

        app.get('/create_account', function (req, res) //Register page
        {
            res.render('create_account_page', { title: "Account
Creation", pageHeader: "FireMAP Portal Account Creation Page",
footerAboutText: "This is the Create Account Page for the FireMAP
Portal." });

        });

        app.post("/create_account", function (req, res) {

            var salt = hasher.createSalt();

            var user =
                {
                    fName: mysql.escape(req.body.firstName),
                    lName: mysql.escape(req.body.lastName),
                    email: mysql.escape(req.body.userEmail),
                    passwordHash:
hasher.computeHash(mysql.escape(req.body.userPassword), salt),
                    salt: salt,
                    orgName: mysql.escape(req.body.orgName),
                    federal: mysql.escape(req.body.federal)
                };

            data.addUser(user, function (result)
            {
                var results = result;
                if (results) //User already exists
                {
                    var passString = encodeURIComponent(user.email+ "
already exists as an account.");
                    res.redirect("/login?valid=" + passString);
                }
                else
                {
                    res.redirect("/training_data_selector");
                }
            });


        });
    };

})(module.exports);
```

controllers

upload_imagery_controller.js

```
(function (uploadImageryController) {

    var auth = require("../auth");

    uploadImageryController.init = function (app)
    {
        app.get('/image_uploader', auth.ensureAuthenticated, function
(req, res) //Download Imagery page
        {
            var userName = "";
            if (req.user) {
                userName = req.user[0].CONTACT_FIRST;
            }
            var aboutMessage = "This page allows you to upload imagery
to our database so it can be classified.";
            res.render('upload_imagery', { title: "Image Uploader",
pageHeader: "Upload Imagery", footerAboutText: aboutMessage, user:
userName });
            res.end();
        });
    };
})(module.exports);
```

download_imagery_controller.js

```
(function (downloadImageryController) {

    var auth = require("../auth");

    downloadImageryController.init = function (app)
    {
        app.get('/download_imagery', auth.ensureAuthenticated, function
(req, res) //Download Imagery page
        {
            var userName = "";
            if (req.user) {
                userName = req.user[0].CONTACT_FIRST;
            }
            var aboutMessage = "This page allows you to download drone
imagery from our database to store locally, train on, or hang up in
your house!";
            res.render('download_imagery', { title: "Download Imagery",
pageHeader: "Download Imagery", footerAboutText: aboutMessage, user:
userName });
            res.end();
        });
    };
})(module.exports);
```

home_controller.js

```
(function (homeController) {

    homeController.init = function (app)
    {
        app.get('/', function (req, res) //Home Page (About Page)
        {
            var userName = "";
            if (req.user)
            {
                userName = req.user[0].CONTACT_FIRST;
            }
            res.render('index', { title: "About FireMAP", pageHeader:
"Home Page", footerAboutText: "This is the Home Page for the FireMAP
Portal.", user: userName});
            res.end();
        });
    };
})(module.exports);
```

index.js

```
(function (controllers)
{
    var HomeController = require("./home_controller");
    var TrainingDataSelector =
require("./training_data_selector_controller");
    var LogoutController = require("./logout_controller");
    var DownloadImageryController =
require("./download_imagery_controller");
    var UploadImageryController =
require("./upload_imagery_controller");
    var ReportsController = require("./reports_controller");

    controllers.init = function (app) {
        HomeController.init(app);
        TrainingDataSelector.init(app);
        LogoutController.init(app);
        DownloadImageryController.init(app);
        UploadImageryController.init(app);
        ReportsController.init(app);
    };
})(module.exports);
```

logout_controller.js

```
(function (logoutController) {

    logoutController.init = function (app) {
        app.post("/logout", function (req, res, next)
        {
            req.session.destroy(function (err) {
                if (err) {
                    console.log("Cannot logout");
                    next(err);
                }
                else
                {
                    var passString = encodeURIComponent("Successfully
logged out.");
                    res.redirect("/login?valid=" + passString);
                }
            });
        });
    };
})(module.exports);
```

reports_controller.js

```
(function (reportsController) {

    var auth = require("../auth");

    reportsController.init = function (app)
    {
        app.get('/reports', auth.ensureAuthenticated, function (req,
res) //Download Imagery page
        {
            var userName = "";
            if (req.user) {
                userName = req.user[0].CONTACT_FIRST;
            }
            var aboutMessage = "This page will allow you to download
and view the reports generated by FireMAP.";
            res.render('reports', { title: "Reports", pageHeader:
"Reports", footerAboutText: aboutMessage, user: userName });
            res.end();
        });
    };
})(module.exports);
```

training_data_selector_controller.js

```
(function (trainingDataSelectorController) {

    const mkdirp = require('mkdirp');
    const fs = require('fs');
    const os = require('os');
    const execFile = require('child_process').execFile;
    const archiver = require('archiver');
    var auth = require("../auth");

    trainingDataSelectorController.init = function (app)
    {
        app.get('/training_data_selector', auth.ensureAuthenticated,
function (req, res) //Training Data Selector page
        {
            var userName = "";
            if (req.user) {
                userName = req.user[0].CONTACT_FIRST;
            }
            var aboutMessage = "The Training Data Application allows
the user to select the data that our classification system will use to
categorize the user's inputs.";
            res.render('training_data_selector', { title: "Training
Data Selector", pageHeader: "Training Data Selector", footerAboutText:
aboutMessage, user: userName });
            res.end();
        });

        app.post("/trainingData", function (req, res)
        {
            var drawingData = req.body.drawingData;
            var fileName = req.body.drawingDataFileName;
            var percentTesting = req.body.percentTesting;
            var pixelGap = req.body.pixelGap;
            var deleteDuplicates = req.body.deleteDuplicates;
            drawingData = drawingData.replace(/\\n/g, os.EOL);
            var filePath = ".\\" + "\\coordinate_selector\\" +
fileName;
               //Create folder for the drawing data and image
            mkdirp(filePath, function (err)
            {
                if (err)
                {
                    console.error(err + "Failed to create file
directory");
                }
                else
                {
                    console.log("Successfully created " + fileName + "
at " + __dirname + filePath);
                }
            });
            //Generate training data with Coordinate Selector
Application
            fs.writeFile(filePath + "\\" + fileName + ".csv",
drawingData, function (err)
            {
                if (err)
                {
                    console.log(err);
                }

                var coordinateSelectorPath = ".\\" +
"\\coordinate_selector\\FireMAPCoordinateSelector.out";
                var inputPath = filePath + "\\" + fileName + ".csv";
                var child = execFile(coordinateSelectorPath,
[inputPath, percentTesting, pixelGap, deleteDuplicates], function
(error, stdout, stderr) {
                    console.log("\nHere is the complete output of the
program:");
                    console.log(stdout);
                    filePath = __dirname;
                    filePath = filePath.replace(/controllers/gi,
'coordinate_selector');

                    var archive = archiver.create('zip', {});
                    var output = fs.createWriteStream(filePath + '/' +
fileName + '.zip');

                    output.on('close', function () {
                        console.log("Finished zipping " + fileName);
                        var downloadFile = filePath + '/' + fileName +
'.zip';

                        res.download(downloadFile);
                    });

                    // good practice to catch this error explicitly
                    archive.on('error', function (err) {
                        throw err;
                    });

                    archive.pipe(output);
                    archive.directory(filePath + '/' + fileName,
fileName);

                    archive.finalize();

                });

            });

        });


    };
})(module.exports);
```

FireMAPCoordinateSelector

```
// FireMAP Coordinate Selector.cpp : Defines the entry point for the
console application.
//

#include "stdafx.h"
#include <fstream>
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>

const char POINT = '1';
const char PENCIL = '2';
const char LINE = '3';
const char POLYLINE = '4';
const char POLYGON = '5';
const char CIRCLE = '6';
const char FLOOD_FILL = '7';
const char AUTO_CLUSTER = '8';

struct point
{
    int x;
    int y;
    std::string label;
};

int intUserInputValidation()//Check to see if user inputted ints are
correctly inputted
{
    //Code for this was gathered from multiple websites and the intro
to c++ book but what finally
    //got it to work was this website:
http://stackoverflow.com/questions/20287186/how-to-check-if-the-input-
is-a-valid-integer-without-any-other-chars
    int number;
    bool quit = false;
    while (quit == false)
    {
        std::cin >> number;
        //If it is the correct data type, and we aren't at the end of
the data type (1234gh, 123jk24 would not pass, but 1234 would)
        if (!std::cin.fail() && (std::cin.peek() == EOF ||
std::cin.peek() == '\n'))
        {
            if (0 > number || number > 100) //Make sure int is between
0 and 100, if not, retry
            {
                std::cout << "\nError. Value must be a whole number
between 0 and 100 Please try again.\n";
                std::cout << "Number: ";
                continue;
            }
            quit = true; //Everything is good, get out
        }
        else
        {
            std::cin.clear(); //clears bad cin
            std::cin.ignore(std::numeric_limits<int>::max(), '\n');
//ignores bad c to the max cin size.
            std::cout << "\nError. Value must be a whole number between
0 and 100 Please try again.\n";
            std::cout << "\nPercent set aside for testing: ";
        }
    }
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<char>::max(), '\n');//ignore
the \n so getline will function
    return number; //Allow main to continue, now knowing the user input
number
}
int stringToInt(std::string intValue)
{
    char *endp;
    int coordinate = std::strtol(intValue.c_str(), &endp, 10);
    if (endp == intValue.c_str())
    {
        //Conversion failed completely, first value is not an int
        return -1; //return negative value to let other function know
an error occurred.
    }
    else if (*endp != 0 && *endp != '.' && *endp != ',') //Fires when
the first part of intValue is an int but the end of it has non-int
values.
    {
        /*The decimal place is in the if statement because js file can
return doubles(training data app), and the decimal place will stop it
here, don't want that
          The comma is in the if statement because if the csv file was
edited with excel there is a chance extra commas got tacked onto the
end of the file
          So even if the string is legit, the excess commas will
prevent it from registering as valid. This ignores commas so it will
still pass as valid.*/
        return -1; //return negative value to let other function know
an error occurred.
    }
    else
    {
        //Entire string was an int value;
        return coordinate; //return correct value
    }
}

std::string grabLabel(std::string * shapeData)
{
    std::string label;
    unsigned int shapeDataCharCount = 0; //Allows us to remove data
from shapeData after for loop ends
    for (shapeDataCharCount = 0; shapeDataCharCount <
(*shapeData).length(); shapeDataCharCount++)
    {
        if ((*shapeData)[shapeDataCharCount] == ',') //If we've come to
our first comma
        {
            shapeDataCharCount++; //Adjust count so we remove the comma
as well from shapeData and not just the char before the comma
            break;
        }
        label += (*shapeData)[shapeDataCharCount];
    }
    (*shapeData) = (*shapeData).substr(shapeDataCharCount); //Remove
shape label from shapeData so we have less to process later.
    if (label.length() != 0)
    {
        return label;
    }
    else //There is no second cell in the csv file (thus no label data)
    {
        //return error message $ signs are here because in training app
data, the user cannot use the $ as a label, thus reducing the chance
the user meant this as an actual label
        return "\n$Error. No label was found in the file.$\n";
    }
}
void removeUnnecessaryData(std::string * shapeData)
{
    unsigned int shapeDataCharCount = 0; //Allows us to remove data
from shapeData after for loop ends
    for (shapeDataCharCount = 0; shapeDataCharCount <
(*shapeData).length(); shapeDataCharCount++)
    {
        if ((*shapeData)[shapeDataCharCount] == ',') //If we've come to
our first comma
        {
            shapeDataCharCount++; //Adjust count so we remove the comma
as well from shapeData and not just the char before the comma
            break;
        }
    }
    (*shapeData) = (*shapeData).substr(shapeDataCharCount); //Remove
unnecessary data from shapeData so we have less to process later.
}
int grabCoordinate(std::string * shapeData, int removeShapeData)
{
    int intCoordinate = -1; //If no changes are made, will stay -1 and
trigger and error message on return
    std::string stringCoordinate; //value that will hold the string,
then we'll convert string to int
    unsigned int shapeDataCharCount = 0; //Allows us to remove data
from shapeData after for loop ends
    if (((*shapeData) == "" || (*shapeData)[0] == ',') &&
removeShapeData != 1) //If its empty
    {
        return -5; //Return this so we know that we were given a blank
string and to just have this value = previous value (aka x2 = x1)
    }
    for (shapeDataCharCount = 0; shapeDataCharCount <
(*shapeData).length(); shapeDataCharCount++)
```

```
    {
        if ((*shapeData)[shapeDataCharCount] == ',') //If we've come to
our first comma
        {
            shapeDataCharCount++; //Adjust count so we remove the comma
as well from shapeData and not just the char before the comma
            if (removeShapeData == 3) //Allows us to ignore previous
value, and grab the correct one (Useful for pencil and polyline, as
they contain a list of points and not just starts/ends)
            {
                shapeDataCharCount--; //Put count back were it was, for
loop will remove comma for us
                stringCoordinate = "";
                removeShapeData = 2;
                continue;
            }
            break;
        }

        stringCoordinate += (*shapeData)[shapeDataCharCount];
        if ((*shapeData)[shapeDataCharCount + 1] == '\0') //If we are
grabbing the last character in our shapeData string
        {
            (*shapeData) = ""; //Make sure string is empty so while
loop in calling function can end properly
            break;
        }
    }
    if (removeShapeData == 1) //If we need to remove the data we just
processed from our string. Useful for lines as we need to keep the end
points for processing the start points next time
    {
        if ((*shapeData).length() == 0)
        {
            //No need to update shapeData string as it is already empty
        }
        else
        {
            (*shapeData) = (*shapeData).substr(shapeDataCharCount);
//Remove unnecessary data from shapeData so we have less to process
later.
        }
    }
    if (removeShapeData == 2 && intCoordinate == -1) //If intCoordinate
didn't get a number and this might be the end of the file ->Most
likeily end of file
    {
        if ((*shapeData)[0] == ',') //If we have a comma in shapeData -
> More evidence of end of file
        {
            return -2; // -1 is used as an error code, -2 will tell us
its the end of the file
        }
    }
    intCoordinate = stringToInt(stringCoordinate); //Convert string to
int and make sure the string was an int
    return intCoordinate; //Return value back to correct shape function
```

```
}
int * grabLineCoordinates(int startX, int startY, int endX, int endY,
int pixelGap) //Grabs every point on the x line and y line and then
puts them into 1 array, removing duplicates
{
    //Because we want a proportionate pixelGap between lines and areas,
we're going to adjust user pixelGap to make it smaller for lines.
    pixelGap = int(std::sqrt(pixelGap)); //Should be a good size. But
may need adjusting later
    if (pixelGap == 0) //In case int rounding causes us to be 0
    {
        pixelGap = 1;
    }
    int tempPoint;
    int *horizontalArray = NULL; //The more horizontal the line  is,
the more points will be in horizontalArray
    int *verticalArray = NULL;  //The more vertical the line  is, the
more points will be in verticalArray
    int *noDuplicatesArray = NULL; //This array will contain points
from both horizontalArray and verticalArray without doubling up from
both. Though duplicates may exist from 1 of the arrays
    if (startX > endX) //If startX is larger than endX, we need to flip
to make for loop work
    {
        //Swap x's
        tempPoint = endX;
        endX = startX;
        startX = tempPoint;
        //Swap y's
        tempPoint = endY;
        endY = startY;
        startY = tempPoint;
    }
    float slope;
    if (startX - endX == 0)
    {
        slope = 0.0; //Can't divide by 0, so slope is just 0
    }
    else
    {
        slope = (float)(startY - endY) / (float)(startX - endX);
//Calculate slope
    }
    float yIntercept = startY - (startX * slope); //Calculate y-
Intercept
    horizontalArray = new int[(2 *(endX - startX)) + 2]; //Create array
that will hold every point that the distance between startX and endX
can grab, plus the start and end points
    int horizontalArrayLength = (2 * (endX - startX)) + 2; //Length of
the array with both endpoints, used to create noDuplicate array
    for (int i = 0; i < horizontalArrayLength; i++)
    {
        horizontalArray[i] = -1;
    }
    for (int i = 0, start = startX; i <= (2 *(endX - startX)); i++)
//Covers all points if line is horizontal
    {
        horizontalArray[i] = start; //X coordinate on our line
        i++; //Move to next location in array
        horizontalArray[i] = (int)((start * slope) + yIntercept);
//Will lose data by down casting to int, worse case scenario we grab a
pixel right next to the line the user drew.
        start++;
    }
    if (startY > endY) //If startY is larger than endY, we need to flip
to make for loop work
    {
        //Swap x's
        tempPoint = endX;
        endX = startX;
        startX = tempPoint;
        //Swap y's
        tempPoint = endY;
        endY = startY;
        startY = tempPoint;
        //Redo line math
        if (startX - endX == 0)
        {
            slope = 0.0; //Can't divide by 0, so slope is just 0
        }
        else
        {
            slope = (float)(startY - endY) / (float)(startX - endX);
///Recalculate slope - only need to do if y start/end points changed
        }
        yIntercept = startY - (startX * slope); //Recalculate y-
Intercept - only need to do if y start/end points changed
    }

    verticalArray = new int[(2 * (endY - startY)) + 2];//Create array
that will hold every point that the distance between startY and endY
can grab plus the start and end points
    int verticalArrayLength = (2 * (endY - startY)) + 2; //Length of
the array with both endpoints, used to create noDuplicate array
    for (int i = 0; i < verticalArrayLength; i++) //Set to default
value
    {
        verticalArray[i] = -1;
    }
    for (int i = 0, start = startY; i <= (2 * (endY - startY)); i++)
//Covers all points if line is vertical
    {
        if (slope != 0)
        {
            verticalArray[i] = (int)((start - yIntercept) / slope);
//Will lose data by down casting to int, worse case scenario we grab a
pixel right next to the line the user drew.
        }
        else //Can't divide by zero, so here is our x
        {
            if (horizontalArrayLength <= 2) //If startX - endX = 0
            {
                verticalArray[i] = startX;
            }
            else
            {
                verticalArray[i] = (int)(start - yIntercept);
            }

        }
        i++;
        verticalArray[i] = start;
        start++;
    }
    noDuplicatesArray = new int[(verticalArrayLength +
horizontalArrayLength) + 1]; //Array that will hold all the values on
the line with no duplicates

    //Code to combine arrays with no duplicates here
    int noDuplicatesArrayCounter = 1; //Start at 1 because element 0
will contain our array length
    noDuplicatesArray[0] = (verticalArrayLength +
horizontalArrayLength) + 1; //Put the length of the array in the first
element so it can have easy access outside this function
    if (horizontalArrayLength > 2) //If startX and endX are the same
value, we only need to collect from verticalArrayLength
    {
        int takenDueToPixelGap = 0;
        for (int i = 0; i < horizontalArrayLength; i++)
        {
            if (horizontalArray[i] == -1)
            {
                break; //Get out of loop, at the end of real data in
horizontalArray
            }
            for (int j = 0; j < verticalArrayLength; j++)
            {
                if (verticalArray[j] == -1)
                {
                    break; //Get out of loop, at the end of real data
in verticaArray
                }
                //If x's match and if even j and i because if even then
we are checking x, and make sure its a legit value
                if (horizontalArray[i] == verticalArray[j] && (j%2 == 0
&& i%2 == 0))
                {
                    i++; j++;
                    if (horizontalArray[i] == verticalArray[j]) //if
y's match -> same coordinate, found a duplicate
                    {
                        //Set to -1 so when we loop through the
verticalArray next, we will be able to easily establish which ones in
the verticalArray are duplicates
                        verticalArray[j-1] = -1;// x = -1
                        verticalArray[j] = -1; // y = -1
                    }
                    i--; //Place counters where they were before
checking y coordinate
                }
            }
            if (takenDueToPixelGap % pixelGap == 0) //Will only add
some pixels to our array. How many we add depends on user specs. If
user enters a 3, we will grab a pixel for every 3 pixels
            {
                noDuplicatesArray[noDuplicatesArrayCounter] =
horizontalArray[i]; //Add x
                noDuplicatesArrayCounter++; //increment counter as
noDuplicates array is an int array not a struct array like
horizontalArray and verticalArray are
                i++; //Move array
                noDuplicatesArray[noDuplicatesArrayCounter] =
horizontalArray[i]; //Add y
                noDuplicatesArrayCounter++; //Increment counter again
            }
            else
            {
                i++; //Move array
            }
            takenDueToPixelGap++;
        }
    }
    if (verticalArrayLength > 2) //if startY and endY are the same
value, we only need to collect from horizontalArray
    {
        int takenDueToPixelGap = 0;
        for (int i = 0; i < verticalArrayLength; i++)
        {
            if (verticalArray[i] != -1 && i%2 == 0) //If a not a
duplicate in the horizontalArray and we're looking at an even digit (x)
            {
                if (takenDueToPixelGap % pixelGap == 0)
                {
                    noDuplicatesArray[noDuplicatesArrayCounter] =
verticalArray[i]; //Add x
                    noDuplicatesArrayCounter++; //Increment counter
                    i++; //move to next spot in array
                    noDuplicatesArray[noDuplicatesArrayCounter] =
verticalArray[i]; //Add y
                    noDuplicatesArrayCounter++; //Increment counter
                }
                else
                {
                    i++; //Move to next spot in array
                }
                takenDueToPixelGap++;
            }
        }
    }
    if (verticalArrayLength == 2 && horizontalArrayLength == 2) //If
startX == endX AND startY == endY, then we only have 1 point,
    {
        //Push the one and only point we have on this line to our
array. Spot [0] is still array size
        noDuplicatesArray[1] = startX;
        noDuplicatesArrayCounter++;
        noDuplicatesArray[2] = startY;
        noDuplicatesArrayCounter++;
    }
    if (noDuplicatesArrayCounter < noDuplicatesArray[0]) //If we
haven't hit the end of our array because we found duplicates
    {
        noDuplicatesArray[noDuplicatesArrayCounter] = -1; //Set the
"last" element in our array to -1 so we know when to stop outside of
this function
    }
    //Delete vertical and horizontal arrays here
    delete[] horizontalArray;
    horizontalArray = NULL;
    delete[] verticalArray;
    verticalArray = NULL;
    return noDuplicatesArray;
}

int intersection(int x1, int y1, int x2, int y2, int x3, int y3, int
x4, int y4)
{
    //Function created based off the function found here:
http://flassari.is/2008/11/line-line-intersection-in-cplusplus/
    struct point
    {
        float x;
        float y;
    };
    point p1 = { (float)x1, (float)y1 }; //Convert ints to floats for
more accuracy //start of ray
    point p2 = { (float)x2, (float)y2 }; //Convert ints to floats for
more accuracy //end of ray (point we are testing)
    point p3 = { (float)x3, (float)y3 }; //Convert ints to floats for
more accuracy //start of polygon line that we test against
    point p4 = { (float)x4, (float)y4 }; //Convert ints to floats for
more accuracy //end of polygon line that we test against

    float d = (p1.x - p2.x) * (p3.y - p4.y) - (p1.y - p2.y) * (p3.x -
p4.x);
    if (d == 0) //No intersection occurred
    {
        return 0;
    }
    //Get position of intersection
    float pre = (p1.x * p2.y - p1.y * p2.x);
    float post = (p3.x * p4.y - p3.y * p4.x);
    float x = (pre * (p3.x - p4.x) - (p1.x - p2.x) * post) / d;
    float y = (pre * (p3.y - p4.y) - (p1.y - p2.y) * post) / d;
    //Check is the x and y coordinates are within both lines
    if (x < std::min(p1.x, p2.x) || x > std::max(p1.x, p2.x) || x <
std::min(p3.x, p4.x) || x > std::max(p3.x, p4.x)) //X failed
    {
        return 0;
    }
    if (y < std::min(p1.y, p2.y) || y > std::max(p1.y, p2.y) || y <
std::min(p3.y, p4.y) || y > std::max(p3.y, p4.y)) //Y failed
    {
        return 0;
    }
    if ((y >= p3.y) && (y >= p4.y)) //if the line we're in contact with
doesn't have a point below the collision point
    {
        //return no collision because we hit a vertex and only want to
count 1 of the 2 lines (the 1 that has a point below our point)
        return 0;
    }
    //Since we made it this far, there has been an intersection between
our two lines and if it is an intersection at a vertex, it "collided"
with the lower line (higher y)
    return 1;
}

bool pointCreator(std::string shapeData, std::vector<point>
&listOfPoints)
{
    //A point contains a shapeID (already removed by this point),
label, color, x, and y.
    std::string label = grabLabel(&shapeData);
    if (label == "\n$Error. No label was found in the file.$\n") //If
label is our error, techically this could trigger if the user entered
this as their label. But the chances are tiny.
    {
        return false;
    }
    removeUnnecessaryData(&shapeData); //This one would remove the
color of the point
    int x = grabCoordinate(&shapeData, 1); //Grab x and remove from
shapeData
    if (x < 0) //No pixel coordinate can be less than 0, therefore, if
x < 0, we found an error in the file and need to stop.
    {
        return false;
    }
    int y = grabCoordinate(&shapeData, 1); //Grab y and remove from
shapeData
    if (y < 0)
    {
        return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
    }
    //Add label, x, and y to our list of coordinates and their label.

    listOfPoints.push_back({ x, y, label }); //Push point to our vector

    return true; //If we've made it this far, the data we have is good
data
}
bool pencilCreator(std::string shapeData, int pixelGap,
std::vector<point> &listOfPoints)
{
    //A pencil contains a shapeID (already removed by this point),
label, color, startX, startY, endX, endY.
    std::string label = grabLabel(&shapeData);
    if (label == "\n$Error. No label was found in the file.$\n") //If
label is our error, technically this could trigger if the user entered
this as their label. But the chances are tiny.
    {
        return false;
    }
    removeUnnecessaryData(&shapeData); //This one would remove the
color of the pencil
    while (shapeData.length() != 0) //Keep running through until we get
to the end of our drawing.
    {
        bool onePointDrawing = false;
        int startX = grabCoordinate(&shapeData, 1); //Grab startX and
remove from shapeData -> 1 allows us to grab first value and delete
        if (startX < 0) //No pixel coordinate can be less than 0,
therefore, if x < 0, we found an error in the file and need to stop.
        {
            return false;
        }
        int startY = grabCoordinate(&shapeData, 1); //Grab startY and
remove from shapeData -> 1 allows us to grab first value and delete
        if (startY < 0)
        {
            return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
        }
        int endX = grabCoordinate(&shapeData, 2); //Grab endX and keep
in shapeData -> 2 allows us to grab first value but not delete
        if (endX < 0) //No pixel coordinate can be less than 0,
therefore, if x < 0, we found an error in the file and need to stop.
        {
            if (endX == -2) //if end of file
            {
                return true; //Get out of here as we have finished the
file (more specifically, have finished the shape)
            }
            else if (endX == -5)
            {
                onePointDrawing = true;
                endX = startX; //Shape data is empty, just give old
value to complete data set
            }
            else
            {
                return false;
            }

        }
        int endY = grabCoordinate(&shapeData, 3); //Grab endY and keep
in shapeData -> 3 allows us to grab second value but not delete
        if (endY < 0)
        {
            if (endY == -5)
            {
                endY = startY; //Shape data is empty, just give old
value to complete data set
            }
            else
            {
                return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
            }
        }

        int *coordinateList = grabLineCoordinates(startX, startY, endX,
endY, pixelGap);

        //Go through coordinateList and create new nodes based on x,y
of coordinate list and label generated at beginning of function
        for (int i = 1; i < coordinateList[0]; i++) //Loop through all
coordinates, creating nodes until we've got them all. Start at 1
because first element is our size
        {
            if (coordinateList[i] <= -1) //If we're at the point in our
array where our data has run out but length hasn't
            {
                break; //At the end of our actual data, get out
            }
            else //Add to node section
            {
                i++; //Increment counter in order to grab y coordinate
                if (coordinateList[i] <= -1) //If we're at the point in
our array where our data has run out but length hasn't (checking y
coordinate)
                {
                    //Error message because there should be a y
coordinate here
                    std::cout << "\nError. We have encountered bad data
with our pencil drawing calculations.\nThere is an X value without a
corresponding Y value";
                    return false; //We're done
                }
                else
                {
                    listOfPoints.push_back({ coordinateList[i-1],
coordinateList[i], label }); //Push point that was on the pencil line
to our vector
                }
            }
        }
        if (onePointDrawing) //Gets triggered if csv file had only 1
point for a point by some rare chance of user drawing a 1 pixel line.
        {
            return true;
        }
    }


    return true; //If we've made it this far, the data we have is good
data
}
bool lineCreator(std::string shapeData, int pixelGap,
std::vector<point> &listOfPoints)
{
    //A line contains a shapeID (already removed by this point), label,
color, startX, startY, endX, endY.
    std::string label = grabLabel(&shapeData);
    if (label == "\n$Error. No label was found in the file.$\n") //If
label is our error, technically this could trigger if the user entered
this as their label. But the chances are tiny.
    {
        return false;
    }
    removeUnnecessaryData(&shapeData); //This one would remove the
color of the line
    while (shapeData.length() != 0) //Keep running through until we get
to the end of our drawing.
    {
        bool onePointDrawing = false;
        int startX = grabCoordinate(&shapeData, 1); //Grab startX and
remove from shapeData -> 1 allows us to grab first value and delete
        if (startX < 0) //No pixel coordinate can be less than 0,
therefore, if x < 0, we found an error in the file and need to stop.
        {
            if (shapeData[0] == ',') //If we are looking at commas
which implies we're at the end of the shape, therefore no error
happened
            {
                return true; //
            }
            return false;
        }
        int startY = grabCoordinate(&shapeData, 1); //Grab startY and
remove from shapeData -> 1 allows us to grab first value and delete
        if (startY < 0)
        {
            return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
        }
        int endX = grabCoordinate(&shapeData, 2); //Grab endX and
remove from shapeData -> 1 allows us to grab first value and delete
        if (endX < 0)
        {
            if (endX == -5)
            {
                onePointDrawing = true;
                endX = startX; //Shape data is empty, just give old
value to complete data set
            }
            else
            {
                return false; //No pixel coordinate can be less than 0,
therefore, if x < 0, we found an error in the file and need to stop.
            }
        }
        int endY = grabCoordinate(&shapeData, 2); //Grab endY and
remove from shapeData -> 1 allows us to grab first value and delete
        if (endY < 0)
        {
            if (endY == -5)
            {
                endY = startY; //Shape data is empty, just give old
value to complete data set
            }
            else
            {
                return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
            }
        }

        int *coordinateList = grabLineCoordinates(startX, startY, endX,
endY, pixelGap);

        //Go through coordinateList and create new nodes based on x,y
of coordinate list and label generated at beginning of function
        for (int i = 1; i < coordinateList[0]; i++) //Loop through all
coordinates, creating nodes until we've got them all. Start at 1
because first element is our size
        {
            if (coordinateList[i] <= -1) //If we're at the point in our
array where our data has run out but length hasn't
            {
                break; //At the end of our actual data, get out
            }
            else //Add to node
            {
                i++; //Increment counter in order to grab y coordinate
                if (coordinateList[i] <= -1) //If we're at the point in
our array where our data has run out but length hasn't (checking y
coordinate)
                {
                    //Error message because there should be a y
coordinate here
                    std::cout << "\nError. We have encountered bad data
with our pencil drawing calculations.\nThere is an X value without a
corresponding Y value";
                    return false; //We're done
                }
                else
                {
                    listOfPoints.push_back({ coordinateList[i - 1],
coordinateList[i], label }); //Push points that were on the line to our
vector
                }
            }
        }
        if (onePointDrawing) //Gets triggered if csv file had only 1
point for a point by some rare chance of user drawing a 1 pixel line.
        {
            return true;
        }
    }


    return true; //If we've made it this far, the data we have is good
data
}
```

```
bool polylineCreator(std::string shapeData, int pixelGap,
std::vector<point> &listOfPoints)
{
    //A polyline contains a shapeID (already removed by this point),
label, color, startX, startY, endX, endY.
    std::string label = grabLabel(&shapeData);
    if (label == "\n$Error. No label was found in the file.$\n") //If
label is our error, technically this could trigger if the user entered
this as their label. But the chances are tiny.
    {
        return false;
    }
    removeUnnecessaryData(&shapeData); //This one would remove the
color of the polyline
    while (shapeData.length() != 0) //Keep running through until we get
to the end of our drawing.
    {
        bool onePointDrawing = false;
        int startX = grabCoordinate(&shapeData, 1); //Grab startX and
remove from shapeData -> 1 allows us to grab first value and delete
        if (startX < 0) //No pixel coordinate can be less than 0,
therefore, if x < 0, we found an error in the file and need to stop.
        {
            return false;
        }
        int startY = grabCoordinate(&shapeData, 1); //Grab startY and
remove from shapeData -> 1 allows us to grab first value and delete
        if (startY < 0)
        {
            return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
        }
        int endX = grabCoordinate(&shapeData, 2); //Grab endX and keep
in shapeData -> 2 allows us to grab first value but not delete
        if (endX < 0) //No pixel coordinate can be less than 0,
therefore, if x < 0, we found an error in the file and need to stop.
        {
            if (endX == -2) //if end of file
            {
                return true; //Get out of here as we have finished the
file (more specifically, have finished the shape)
            }
            else if (endX == -5)
            {
                onePointDrawing = true;
                endX = startX;
            }
            else
            {
                return false;
            }
        }
        int endY = grabCoordinate(&shapeData, 3); //Grab endY and keep
in shapeData -> 3 allows us to grab second value but not delete
        if (endY < 0)
        {
            if (endY == -5)
            {
                endY = startY; //Shape data is empty, just give old
value to complete data set
            }
            else
            {
                return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
            }
        }

        int *coordinateList = grabLineCoordinates(startX, startY, endX,
endY, pixelGap);

        //Go through coordinateList and create new nodes based on x,y
of coordinate list and label generated at beginning of function
        for (int i = 1; i < coordinateList[0]; i++) //Loop through all
coordinates, creating nodes until we've got them all. Start at 1
because first element is our size
        {
            if (coordinateList[i] <= -1) //If we're at the point in our
array where our data has run out but length hasn't
            {
                break; //At the end of our actual data, get out
            }
            else //Add to node
            {
                i++; //Increment counter in order to grab y coordinate
                if (coordinateList[i] <= -1) //If we're at the point in
our array where our data has run out but length hasn't (checking y
coordinate)
                {
                    //Error message because there should be a y
coordinate here
                    std::cout << "\nError. We have encountered bad data
with our polyline drawing calculations.\nThere is an X value without a
corresponding Y value";
                    return false; //We're done
                }
                else
                {
                    listOfPoints.push_back({ coordinateList[i - 1],
coordinateList[i], label }); //Push point that was on the polyline to
our vector
                }
            }
        }
        if (onePointDrawing) //Gets triggered if csv file had only 1
point for a point by some rare chance of user drawing a 1 pixel line.
        {
            return true;
        }
    }


    return true; //If we've made it this far, the data we have is good
data
```

```
}
bool polygonCreator(std::string shapeData, int pixelGap,
std::vector<point> &listOfPoints)
{
    //Used these links to help build this function
    //http://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-
inside-a-polygon/
    //http://stackoverflow.com/questions/217578/how-can-i-determine-
whether-a-2d-point-is-within-a-polygon
    //A polygon contains a shapeID (already removed by this point),
label, color, alpha level, startX, startY, endX, endY.
    struct point
    {
        int x;
        int y;
    };
    std::vector<point> polygonVertices;
    std::string label = grabLabel(&shapeData);
    if (label == "\n$Error. No label was found in the file.$\n") //If
label is our error, technically this could trigger if the user entered
this as their label. But the chances are tiny.
    {
        return false;
    }
    removeUnnecessaryData(&shapeData); //This one would remove the
color of the polygon
    removeUnnecessaryData(&shapeData); //This one would remove the fill
color of the polygon
    while (shapeData.length() != 0) //Keep running through until we get
to the end of our drawing.
    {
        int x = grabCoordinate(&shapeData, 1); //Grab startX and remove
from shapeData -> 1 allows us to grab first value and delete
        if (x < 0) //No pixel coordinate can be less than 0, therefore,
if x < 0, we found an error in the file and need to stop.
        {
            if (shapeData[0] == ',') //If excess commas are the reason
we returned an error
            {
                break; //We're at the end of the shape, break out of
the while loop
            }
            return false;
        }
        int y = grabCoordinate(&shapeData, 1); //Grab startY and remove
from shapeData -> 1 allows us to grab first value and delete
        if (y < 0)
        {
            return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
        }
        polygonVertices.push_back({ x, y }); //Add x and y to our
vector
    }
    polygonVertices.push_back({ polygonVertices[0].x,
polygonVertices[0].y }); //Push first X, Y to end as it is our starting
X,Y coordinate AND our ending X,Y coordinate
```

```
    for (unsigned int i = 0; i < polygonVertices.size() - 1; i++)
    {
        int *coordinateList = grabLineCoordinates(polygonVertices[i].x,
polygonVertices[i].y, polygonVertices[i + 1].x, polygonVertices[i +
1].y, pixelGap);
        //Go through coordinateList and create new nodes based on x,y
of coordinate list and label generated at beginning of function
        for (int j = 1; j < coordinateList[0]; j++) //Loop through all
coordinates, creating nodes until we've got them all. Start at 1
because first element is our size
        {
            if (coordinateList[j] <= -1) //If we're at the point in our
array where our data has run out but length hasn't
            {
                break; //At the end of our actual data, get out
            }
            else //Add to node
            {
                j++; //Increment counter in order to grab y coordinate
                if (coordinateList[j] <= -1) //If we're at the point in
our array where our data has run out but length hasn't (checking y
coordinate)
                {
                    //Error message because there should be a y
coordinate here
                    std::cout << "\nError. We have encountered bad data
with our polyline drawing calculations.\nThere is an X value without a
corresponding Y value";
                    return false; //We're done
                }
                else
                {
                    listOfPoints.push_back({ coordinateList[j - 1],
coordinateList[j], label }); //Pushes border points onto polygon
                }
            }
        }
    }

    point max = { polygonVertices[0].x, polygonVertices[0].y };
//Default value for max
    point min = { polygonVertices[0].x, polygonVertices[0].y };
//Default value for min
    //Get minX, maxX, minY, and maxY to create bounding box around
polygon.
    for (unsigned int i = 0; i < polygonVertices.size(); i++)
    {
        if (polygonVertices[i].x < min.x)
        {
            min.x = polygonVertices[i].x;
        }
        else if (polygonVertices[i].x > max.x)
        {
            max.x = polygonVertices[i].x;
        }
        if (polygonVertices[i].y < min.y)
        {
            min.y = polygonVertices[i].y;
        }
        else if (polygonVertices[i].y > max.y)
        {
            max.y = polygonVertices[i].y;
        }
    }
    //adjust bounding box to give some gap space
    min.x = min.x - 1;
    min.y = min.y - 1;
    max.x = max.x + 1;
    max.y = max.y + 1;
    //Shoot lines across bounding box to x,y point, counting
intersections with polygon sides
    std::vector<point> pointsInPolygon; //Vector that will hold all of
the points inside our polygon
    int intersectCount = 0;
    int pointFound = 0;

    //http://alienryderflex.com/polygon/ concept behind this code
    //Basic idea:
    //if odd intersection, point inside polygon
        //Store this point in a vector
    //if even intersection, point outside polygon
    //select next point backed on pixelGap
    //Once pixel is >= to far bounding box line
    //Move shooting line down
    for (int i = min.y; i < max.y; i ++)
    {
        for (int j = min.x; j < max.x; j ++) //j moves left to right in
bounding box
        {
            //Line will be from (min.x, i) to (j, i) vs polygonVertices
            unsigned int verticesCount = 0;
            while (verticesCount < polygonVertices.size() - 1) //Lines
= vertices - 1
            {
                intersectCount += intersection(min.x, i, j, i,
                polygonVertices[verticesCount].x,
polygonVertices[verticesCount].y, polygonVertices[verticesCount + 1].x,
polygonVertices[verticesCount + 1].y);
                //intersectCount holds how many times we have crossed a
polygon line. if odd, inside, if even, outside
                verticesCount++; //Increment count
            }
            if (intersectCount % 2 != 0) //If odd -> inside polygon. If
even -> outside and don't push to vector
            {
                if (pointFound % pixelGap == 0) //If the point found
inside our polygon is the same distance from the last point found as
our pixelGap, add to our vector
                {
                    pointsInPolygon.push_back({ j, i }); //Push point
that was inside the polygon to our vector
                    pointFound = 0; //Reset so we don't overflow with a
large polygon
```

```
                }
                pointFound++;
            }
            verticesCount = 0; //reset count
            intersectCount = 0; //reset count

        }
    }

    for (unsigned int i = 0; i < polygonVertices.size(); i++) //pushes
vertices onto vector
    {
        listOfPoints.push_back({ polygonVertices[i].x,
polygonVertices[i].y, label }); //Push point that was inside the
polygon to our vector
    }
    for (unsigned int i = 0; i < pointsInPolygon.size(); i++) //pushes
points inside polygon onto vector
    {
        listOfPoints.push_back({ pointsInPolygon[i].x,
pointsInPolygon[i].y, label }); //Push point that was inside the
polygon to our vector
    }

    return true; //If we've made it this far, the data we have is good
data
}
bool circleCreator(std::string shapeData, int pixelGap,
std::vector<point> &listOfPoints)
{
    //A point contains a shapeID (already removed by this point),
label, color, fillColor, x, y, and radius.
    struct center
    {
        int x;
        int y;
        int radius;
    };
    std::vector<center> circleData;
    std::string label = grabLabel(&shapeData);
    if (label == "\n$Error. No label was found in the file.$\n") //If
label is our error, techically this could trigger if the user entered
this as their label. But the chances are tiny.
    {
        return false;
    }
    removeUnnecessaryData(&shapeData); //This one would remove the
color of the polygon
    removeUnnecessaryData(&shapeData); //This one would remove the fill
color of the polygon
    int x = grabCoordinate(&shapeData, 1); //Grab startX and remove
from shapeData -> 1 allows us to grab first value and delete
    if (x < 0) //No pixel coordinate can be less than 0, therefore, if
x < 0, we found an error in the file and need to stop.
    {
        return false;//No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
```

```
    }
    int y = grabCoordinate(&shapeData, 1); //Grab startY and remove
from shapeData -> 1 allows us to grab first value and delete
    if (y < 0)
    {
        return false; //No pixel coordinate can be less than 0,
therefore, if y < 0, we found an error in the file and need to stop.
    }
    int radius = grabCoordinate(&shapeData, 1); //Grab startY and
remove from shapeData -> 1 allows us to grab first value and delete
    if (radius < 0)
    {
        return false; //No radius can be less than 0, therefore, if
radius < 0, we found an error in the file and need to stop.
    }
    circleData.push_back({ x, y, radius }); //Add x and y to our vector

    //Grabed code from http://jsperf.com/point-in-circle //Slower than
other versions, but I know this one is accurate (and simple to
intergrate).
    int pointFound = 0;
    for (int x = circleData[0].x - circleData[0].radius; x <=
circleData[0].x + circleData[0].radius; x ++)
    {
        for (int y = circleData[0].y - circleData[0].radius; y <=
circleData[0].y + circleData[0].radius; y ++)
        {
            //If distance from point to center of circle is <= to
radius, we're inside the circle (a^2 + b^2 = c^2)
            if (( ((x - circleData[0].x) * (x - circleData[0].x)) + ((y
- circleData[0].y) * (y - circleData[0].y)) ) <= (circleData[0].radius
* circleData[0].radius))
            {
                if (pointFound % pixelGap == 0) // if we're looking at
the correct pixel based on gap, we can grab it
                {
                    listOfPoints.push_back({ x, y, label }); //Push
point that was inside the circle to our vector
                    pointFound = 0; //reset to make sure we don't get
an overflow in our int if the circle is crazy huge
                }
                pointFound++;
            }
        }
    }


    return true; //If we've made it this far, the data we have is good
data
}
bool autoClusterCreator(std::string shapeData) //Doesn't exist in
Training data app yet
{
    //An auto-cluster contains I have no idea... Isn't built yet
    std::cout << "The auto-cluster shape is still a work in
progress.\n";
    return true;
```

```
}
bool floodFillCreator(std::string shapeData) //Doesn't exist in
training data app yet
{
    //A flood fill contains I have no idea... Isn't built yet
    std::cout << "The flood fill shape is still a work in progress.\n";
    return true;
}

bool coordinateCreator(std::string shapeData, int
percentSetAsideForTesting, int pixelGap, std::vector<point>
&listOfPoints) //Determines what type of shape is in the file and then
calls that shape specific function
{
    std::string shapeID;
    unsigned int shapeDataCharCount = 0;
    for (shapeDataCharCount = 0; shapeDataCharCount <
shapeData.length(); shapeDataCharCount++)
    {
        if (shapeData[shapeDataCharCount] == ',') //If we've come to
our first comma
        {
            shapeDataCharCount++; //Adjust count so we remove the comma
as well from shapeData and not just the char before the comma
            break;
        }
        shapeID += shapeData[shapeDataCharCount];
    }
    shapeData = shapeData.substr(shapeDataCharCount); //Remove shape id
from shapeData so we have less to process later.
    /*Although we grab the data until the first comma, the shapeID
should really be the first char in the string, and only 1 char long
(single digit number, 1- 8)
    Therefore, to save time by not converting data types, we can
compare what our input shape should be (if created with training app),
by doing the following */
    bool validData;
    if (shapeID[0] == POINT) //If shape is a point
    {
        validData = pointCreator(shapeData, listOfPoints);
    }
    else if (shapeID[0] == PENCIL) //if shape is a pencil
    {
        validData = pencilCreator(shapeData, pixelGap, listOfPoints);
    }
    else if (shapeID[0] == LINE) //if shape is a line
    {
        validData = lineCreator(shapeData, pixelGap, listOfPoints);
    }
    else if (shapeID[0] == POLYLINE) //if shape is a polyline
    {
        validData = polylineCreator(shapeData, pixelGap, listOfPoints);
    }
    else if (shapeID[0] == POLYGON) //if shape is a polygon
    {
        validData = polygonCreator(shapeData, pixelGap, listOfPoints);
    }
    else if (shapeID[0] == CIRCLE) //if shape is a circle
    {
        validData = circleCreator(shapeData, pixelGap, listOfPoints);
    }
    else if (shapeID[0] == AUTO_CLUSTER) //if shape is an auto-cluster
    {
        validData = autoClusterCreator(shapeData);
    }
    else if (shapeID[0] == FLOOD_FILL) //if shape is a flood-fill
    {
        validData = floodFillCreator(shapeData);
    }
    else //Invalid data
    {
        std::cout << "\nError. Could not find shape ID. This is most
likely caused by incorrect data in the file.\nPlease choose a different
file.\n";
        return false;
    }
    if (!validData) //If one of the shape functions returned false
(invalid data found in that shape)
    {
        std::cout << "\nError. File contained bad data. Please try a
different file.\n";
        return false;
    }
    return true; //Everything worked
}

void deleteDuplicates(std::vector<point> &listOfPoints)
{
    std::vector<int> positionToDeleteLater;
    for (unsigned int i = 0; i < listOfPoints.size() - 1; i++)
    {
        if (listOfPoints[i].x == listOfPoints[i + 1].x)
        {
            if (listOfPoints[i].y == listOfPoints[i + 1].y)
            {
                if (listOfPoints[i].label == listOfPoints[i + 1].label)
//If same point and label, then a true duplicate, delete only one
                {
                    listOfPoints.erase(listOfPoints.begin() + i + 1);
                }
                else //if same point but different labels, then a false
duplicate, delete both
                {
                    positionToDeleteLater.push_back(i);
                }
            }
        }
    }
    for (unsigned int i = 0; positionToDeleteLater.size(); i++)
    {
        listOfPoints.erase(listOfPoints.begin() +
positionToDeleteLater[i]);
    }
}
```

```
void writePointsToFile(int percentSetAsideForTesting,
std::vector<point> &listOfPoints, std::string fileName)
{
    fileName.erase(fileName.end() - 4, fileName.end()); //Remove the
.csv from the file name and attach it on the end when we create our
file
    std::cout << "\nWriting to "<< fileName << "Training.csv and " <<
fileName << "Testing.csv" << " now...\n\n";
    std::ofstream outputFileTraining, outputFileTesting;
    outputFileTraining.open(fileName + "Training.csv");
    outputFileTesting.open(fileName + "Testing.csv");
    outputFileTraining << "Label, X, Y\n";
    outputFileTesting << "Label, X, Y\n";
    if (percentSetAsideForTesting == 0) //Prevents rand() from grabbing
0 as a value
    {
        percentSetAsideForTesting = -1;
    }
    for (unsigned int i = 0; i < listOfPoints.size(); i++)
    {
        if (rand() % 100 < percentSetAsideForTesting) //Random number,
but because it isn't seeded, it is the same numbers each time program
is restarted
        {
            outputFileTesting << listOfPoints[i].label << "," <<
listOfPoints[i].x << "," << listOfPoints[i].y << "\n";
        }
        else
        {
            outputFileTraining << listOfPoints[i].label << "," <<
listOfPoints[i].x << "," << listOfPoints[i].y << "\n";
        }
    }
    outputFileTesting.close();
    outputFileTraining.close();
}

int main(int argc, char * argv[])
{
    std::ifstream trainingGeometries; // Our input file. This is the
csv file that comes from the Training Data App
    std::string trainingGeometriesFileName;
    bool iQuit = false;
    int percentSetAsideForTesting;
    int pixelGap; //We don't need to grab every pixel inside a shape,
this determines the pixel offset of the ones we do grab
    int deleteDuplicate; //Holds user input to whether or not we delete
duplicates in this program.

    //NOTE: Not sure if point has x, y, and label yet
    std::vector<point> listOfPoints; //Vector that holds x, y, and
label of all our points.

    if (argc < 5) //If not using command line arguments
    {
        std::cout << "Please enter the name of the training geometries
file you would like to load.\n"; //Outside loop so user doesn't get
excess text when entering wrong file name data
        while (!iQuit)
        {
            std::cout << "File Name: ";
            std::cin >> trainingGeometriesFileName;
            if (trainingGeometriesFileName == "Quit" ||
trainingGeometriesFileName == "quit" || trainingGeometriesFileName ==
"q" || trainingGeometriesFileName == "Q")
            {
                return 0; //User wants to quit, so let's return true
            }
            trainingGeometries.open(trainingGeometriesFileName); //open
the file
            while (trainingGeometries.fail())
            {
                trainingGeometries.clear();
                std::cout << "The file name and path you gave us could
not be loaded. Please try again.\n";
                std::cout << "File name: ";
                std::cin >> trainingGeometriesFileName;
                if (trainingGeometriesFileName == "Quit" ||
trainingGeometriesFileName == "quit" || trainingGeometriesFileName ==
"q" || trainingGeometriesFileName == "Q")
                {
                    return 0; //User wants to quit, so let's return
true
                }
                trainingGeometries.open(trainingGeometriesFileName);
//open the file
            }
            if (trainingGeometries.is_open()) //The file given can be
opened
            {
                std::cout << "\nWhat percent of the file would you like
to set aside for testing? We suggest 30.\nPercent set aside for
testing: ";
                percentSetAsideForTesting = intUserInputValidation();
//Validates number, will stay in function until data is correct
                std::cout << "\nWhat distance would you like to have
between pixels? We suggest that you take every 5 pixels.\nPixel gap: ";
                pixelGap = intUserInputValidation(); //Validates
number, will stay in function until data is correct
                pixelGap++; //Add 1 to pixelGap to get accurate user
interpration. Ex. if user enter 0 so they can get every point, x % 1 ==
0 will grab every point.
                std::cout << "\nWould you like this program to delete
duplicate points? Enter 1 or 2.\nYes[1]\tNo[2]: ";
                deleteDuplicate = intUserInputValidation(); //Validates
number, will stay in function until data is correct
                std::string shapeData;
                if (getline(trainingGeometries, shapeData))
                {
                    int numberOfShapes = stringToInt(shapeData);
//Holds how many shapes we have, allows us to use a for loop
                    bool validData;
                    if (numberOfShapes < 0) //if stringToInt returned
bad data
                    {
                        std::cout << "Error. File contained bad data.";
                        validData = false; //We have bad data, will
cause program to restart
                    }
                    for (int i = 0; i < numberOfShapes; i++)
                    {
                        getline(trainingGeometries, shapeData); //Grabs
a new line
                        validData = coordinateCreator(shapeData,
percentSetAsideForTesting, pixelGap, listOfPoints); //gets the x, y,
label from line
                        if (!validData) //If the file does not contain
valid input data
                        {
                            break; //Get out of the for loop, so we can
restart program
                        }
                    }
                    if (!validData) //If the file does not contain
valid input data
                    {
                        std::cout << "\nRestarting program... Please
choose a file with the correct data inside.\n";
                        trainingGeometries.close(); //close file
                        continue; //Restart program
                    }
                }
                trainingGeometries.close(); //Close our file so we can
restart app
                if (deleteDuplicate == 1)
                {
                    deleteDuplicates(listOfPoints);
                }
                writePointsToFile(percentSetAsideForTesting,
listOfPoints, trainingGeometriesFileName); //write the points we
collected to our training and testing files
                listOfPoints.clear(); //Clear the list for if the user
wants to create another list
                std::cout << "We have finished writing your files.\nIf
you would like to quit. Type 'Quit'. If you want to data points from
another file, enter that file path.\n";
            }
            else //The file given can't be opened
            {
                trainingGeometries.close();
                std::cout << "We could not open the file. Please try
again.\n";
                continue;
            }
        }

    }
    else // We are using command line arguments, which means we are
assuming the user knows what they are entering/data types will be
correct as well
    {
        while (!iQuit)
        {
            trainingGeometriesFileName = argv[1];
            trainingGeometries.open(trainingGeometriesFileName); //open
the file
            if (trainingGeometries.is_open()) //The file given can be
opened
            {
                percentSetAsideForTesting = stringToInt(argv[2]);
                pixelGap = stringToInt(argv[3]);
                pixelGap++; //Add 1 to pixelGap to get accurate user
interpration. Ex. if user enter 0 so they can get every point, x % 1 ==
0 will grab every point.
                deleteDuplicate = stringToInt(argv[4]);
                std::string shapeData;
                if (getline(trainingGeometries, shapeData))
                {
                    int numberOfShapes = stringToInt(shapeData);
//Holds how many shapes we have, allows us to use a for loop
                    bool validData;
                    if (numberOfShapes < 0) //if stringToInt returned
bad data
                    {
                        std::cout << "Error. File contained bad data.";
                        validData = false; //We have bad data, will
cause program to restart
                    }
                    for (int i = 0; i < numberOfShapes; i++)
                    {
                        getline(trainingGeometries, shapeData); //Grabs
a new line
                        validData = coordinateCreator(shapeData,
percentSetAsideForTesting, pixelGap, listOfPoints); //gets the x, y,
label from line
                        if (!validData) //If the file does not contain
valid input data
                        {
                            break; //Get out of the for loop, so we can
restart program
                        }
                    }
                    if (!validData) //If the file does not contain
valid input data
                    {
                        std::cout << "\nClosing program... Please
choose a file with the correct data inside.\n";
                        trainingGeometries.close(); //close file
                        return 1; //End program
                    }
                }
                trainingGeometries.close(); //Close our file so we can
restart app
                if (deleteDuplicate == 1)
                {
                    deleteDuplicates(listOfPoints);
                }
                writePointsToFile(percentSetAsideForTesting,
listOfPoints, trainingGeometriesFileName); //write the points we
collected to our training and testing files
                listOfPoints.clear(); //Clear the list for if the user
wants to create another list
                std::cout << "We have finished writing your files. You
will find them where your original csv file is located.\n";
                return 0;
            }
            else //The file given can't be opened
            {
                trainingGeometries.close();
                std::cout << "We could not open the file. Please try
again.\n";
                return 1;
            }
        }
    }

    return 0;
}
```

data

database.js

```
// Database.js

(function (database)
{

    var mysql = require('mysql');
    const myLogMod = require('../server_modules/log.js');

    var connection = mysql.createConnection
        ({
            host: '127.0.0.1',
            user: 'root',
            port: '3306',
            password: 'FireMAP',
            database: 'firemap',
            multipleStatements: true
        });

    database.init = function (app)
    {
        connection.connect(function (err) {
            if (err) {
                myLogMod.error('Error connecting: ' + err.stack);
                return;
            }
            myLogMod.info('Connected as id ' + connection.threadId);
        });
    };

    database.addUser = function(user, callback)
    {
        var queryString = "SELECT addUser(" + user.fName + ", " +
user.lName + ", " + user.email + ", '" + user.passwordHash + "', " +
user.orgName + ", " + user.federal + ", '" + user.salt + "')";
        var results = "";
        connection.query(queryString, function (err, data)
        {
            if (err) throw err;
            var stringData = JSON.stringify(data[0]);
            var myRegex = new RegExp(':-1'); //if -1 then user was
already in the database
            results = myRegex.test(stringData);
            return callback(results);
        });
    };

    database.getUser = function (username, next)
    {
        escapedUsername = mysql.escape(username);
        var queryString = "SELECT * FROM contact WHERE CONTACT_EMAIL =
" + escapedUsername;
        connection.query(queryString, function (err, data) {
            if (err)
            {
                next(err);
            }
            else
            {
                if (data[0])
                {
                    if (escapedUsername === "'" + data[0].CONTACT_EMAIL
+ "'")
                    {
                        next(null, data);
                    }
                    else
                    {
                        next(null, false);
                    }
                }
                else
                {
                    next(null, false);
                }
            }

        });
    };

    connection.on('close', function (err) {
        if (err) {
            //Unexpected closing of connection, reconnect
            connection = mysql.createConnection(connection.config);
        } else {
            console.log('Connection id ' + connection.threadId +'
closed normally.');
        }
    });


})(module.exports);
```

index.js

```
(function (data) {
    var database = require("./database");

    data.init = function (app) {
        database.init(app);
    };
})(module.exports);
```

public

css

create_account.css

```
/*UPDATE 11/16/2015*/

#nudgeContactInfoTitle {
    text-indent: 32px;
}

#nudgeOrgInfoTitle {
    text-indent: 10px;
}

#nudgeButton {
    text-indent: 73px;
}

#nudgeFederal {
    text-indent: 30px;
}




.textBox {
    width: 217px;
    border: 2px solid rgba(175, 175, 175, 0.75);
}

.selectBox {
    width: 222px;
    border: 2px solid rgba(175, 175, 175, 0.75);
}

#toolTipButton {
    position: absolute;
    padding: 2px;
    z-index: 10;
}

.removeBold {
    font-weight: normal;
}


a.toolTip.federal {
    outline: none;
    position: absolute;
    padding-top: 2px;
    padding-left: 41px;
}
```

```
a.toolTip {
    outline: none;
    position: absolute;
    padding-top: 2px;
    padding-left: 2px;
}

    a.toolTip strong {
        line-height: 30px;
    }

    a.toolTip:hover {
        text-decoration: none;
    }

    a.toolTip span {
        z-index: 10;
        display: none;
        padding: 14px 20px;
        margin-top: -30px;
        margin-left: 28px;
        width: 300px;
        line-height: 16px;
    }

    a.toolTip:hover span {
        display: inline;
        position: absolute;
        color: #FFFFFF;
        text-shadow: 1px 1px #000000;
        border: 2px solid #000;
        background: #be0f34;
    }

.callout {
    z-index: 20;
    position: absolute;
    bottom: 41px;
    border: 0;
    left: -12px;
}
.calloutLong {
    z-index: 20;
    position: absolute;
    bottom: 71px;
    border: 0;
    left: -12px;
}
.calloutMiddle {
    z-index: 20;
    position: absolute;
    bottom: 57px;
    border: 0;
    left: -12px;
}
```

```
.callout.organization
{
    z-index: 20;
    position: absolute;
    bottom: 57px;
    border: 0;
    left: -12px;
    }
.calloutLong.organization {
    z-index: 20;
    position: absolute;
    bottom: 57px;
    border: 0;
    left: -12px;
}

a.toolTip span {
    border-radius: 14px;
    box-shadow: 5px 5px 8 px #CCC;
}

.break {
    width: 100%;
    height: 10px;
}

.bottomBreak {
    margin-top: 10px;
    display: inline-block;
    width: 100%;
    height: 10px;
}


/*END_UPDATE 11/16/2015*/
#defaultProvidenceSelector {
    position: absolute;
}

#usaProvidenceSelector {
    position: absolute;
}

#CanadaProvidenceSelector {
    position: absolute;
}

.LV_valid {
    color: rgba(0, 180, 0, 0.75);
}

.LV_invalid {
    color: rgb(240, 0, 0);
}

.LV_validation_message {
    margin: 0 0 0 2px;
}

.LV_valid_field {
    border: 2px solid rgba(0, 180, 0, 0.75);
    width: 217px;
}

.LV_invalid_field {
    border: 2px solid rgba(240, 0, 0, 0.75);
    width: 217px;
}
```

site.css

```
/*site.css*/
/*If you need to the navbar code, go to bootstrap.css in lib*/

a {
    color: #ffffff;
    text-decoration: none;
}

    a:hover {
        color: black;
        text-decoration: none;
        text-shadow: none;
        cursor: pointer;
    }

.header {
    text-align: center;
    color: #ffffff;
    text-shadow: 1px 1px #000000;
    background-color: #be0f34;
    margin-top: 5px;
    margin-right: 5px;
    margin-left: 5px;
}

@media(max-width:400px) {
    .shrinkText {
        font-size: 9px;
    }
}

/*This is a filler class. It takes up blank space to make things align
better*/
.blankSpace {
    background-color: #ffffff;
    text-align: center;
    color: #ffffff;
}
/***********************************************************************
********/

.main {
    height: 600px; /*This height helps determine the @media height for
the footer (aka navbar-fixed-bottom) response inside of the
bootstrap.css file*/
}

.logo {
    height: 100px;
    width: auto;
    padding-top: 2px;
    float: left;
}


.submitButtons {
    margin-bottom: 5px;
    text-indent: -4px;
    color: #000;
    margin-left: 2px;
}

.loadAndDeleteButtons {
    margin-bottom: 5px;
    margin-top: 5px;
    text-indent: -4px;
    color: #000;
}

.toolSelector {
    margin-top: 5px;
    height: 600px;
}

.selector {
    clear: both;
    background-color: #be0f34;
    border-radius: 13px;
    text-align: center;
    color: #ffffff;
    text-shadow: 1px 1px #000000;
}

.selectorButtonRow1 {
    margin-right: 0;
    margin-left: 2%;
    margin-bottom: 5%;
    color: #000;
}

.selectorButtonRow2 {
    margin-right: 1%;
    margin-left: 1%;
    margin-bottom: 2.5%;
    color: #000;
}

.selectorButtonRow3 {
    margin-right: 1.5%;
    margin-left: 1%;
    margin-bottom: 5%;
    color: #000;
}
::-webkit-input-placeholder { /* Chrome/Opera/Safari */
    font-size: 12px;
}

::-moz-placeholder { /* Firefox 19+ */
    font-size: 12px;
}

:-ms-input-placeholder { /* IE 10+ */
    font-size: 12px;
}


:-moz-placeholder { /* Firefox 18- */
    font-size: 12px;
}

.dataLabel {
    width: 68.5%;
    font-size: 18px;
}

.labelWidth
{
    width: 30%;
    height: auto;
}

.infoText {
    font-size: xx-small;
    width: auto;
    text-align: left;
    margin-left: 1px;
    text-shadow: none;
}

.pixelLocations {
    width: 100%;
    text-align: center;
    padding-left: 0;
    padding-right: 0;
}

.labelSelect {
    margin-bottom: 10px;
    width: 100%;
    float: left;
    padding-left: 0;
}

.trainingDataSection {
    padding-top: 75px;
    margin-top: 5px;
    text-align: center;
}

@media(max-width:800px) {
    .trainingDataSection {
        padding-top: 15px;
    }
}

.zoom {
    margin: 0;
    padding: 0;
    clear: none;
}

.floatLeft {
    float: left;
}

.clearRight {
    clear: right;
}

.zoomIcon {
    width: 20px;
    height: 20px;
    clear: none;
    position: relative;
}

#scaleIn {
    padding: 3px;
    font-size: 13px;
    line-height: 1.5px;
    border-radius: 3px;
}

#scaleOut {
    padding: 3px;
    font-size: 13px;
    line-height: 1.5px;
    border-radius: 3px;
}

#scaledImage {
    text-align: center;
    vertical-align: middle;
    overflow: scroll;
}

.grabCursor {
    cursor: grab;
}

.crosshairCursor {
    cursor: crosshair;
}
```

```
.img {
    position: relative;
    z-index: 150;
    pointer-events: auto;
}

#myCanvas {
    border: 1px solid black; /*Might be causing coordinates to get off
by 1 pixel*/
}

#spinner {
    visibility: hidden;
    position: absolute;
    padding-top: 100px;
    width: 95%;
}

#inputImageDiv {
    visibility: hidden;
    width: 200px;
    height: 200px;
    overflow: hidden;
}

#outputImage {
    border: 1px solid black;
    z-index: 100;
}

#image {
   /* padding-top: -75px; */
    clear: both;
}

#upload {
    display: none;
}

.marginRightLeft {
    margin-right: 5px;
}

.Picker {
    margin-top: 5px;
    text-align: center;
}

.colorPickerBackground {
    background-color: #BBB;
    color: #000;
    text-shadow: 1px 1px #fff;
    border-radius: 13px;
    margin-top: 5px;
    clear: both;
}
```

```
.colorRow {
    width: 100%;
}

.marginSpace {
    margin-bottom: 5px;
}

#polylineConfirmation {
    margin-left: 0;
    text-indent: 0;
}

#completePolygon {
    margin-left: 0;
    text-indent: 0;
}
/*Default colors for the color picker. background-color is the color of
the button and border of tools. Color is the transparent color of
background-color and fill of polygon/circle*/
.btn-black, .btn-white, .btn-darkRed, .btn-red, .btn-orange, .btn-
yellow, .btn-neonGreen, .btn-pukeGreen, .btn-lightBlue, .btn-blue,
.btn-purple, .btn-pink {
    margin-left: 1px;
    margin-right: 1px;
    font-size: 9px;
}

.btn-black {
    background-color: black;
    color: rgb(0,0,0);
    color: rgba(0,0,0,.35);
}

.btn-white {
    background-color: white;
    color: rgb(255,255,255);
    color: rgba(255,255,255, .35);
}

.btn-darkRed {
    background-color: darkred;
    color: rgb(139,0,0);
    color: rgba(139,0,0, .35);
}

.btn-red {
    background-color: red;
    color: rgb(255, 0, 0);
    color: rgba(255, 0, 0, .35);
}

.btn-orange {
    background-color: #ffa500;
    color: rgb(255, 165, 0);
    color: rgba(255, 165, 0, .35);
```

```
}

.btn-yellow {
    background-color: yellow;
    color: rgb(255, 255, 0);
    color: rgba(255, 255, 0, .35);
}

.btn-neonGreen {
    background-color: #00FF00;
    color: rgb(0, 255, 0);
    color: rgba(0, 255, 0, .35);
}

.btn-pukeGreen {
    background-color: OliveDrab;
    color: rgb(107, 142, 35);
    color: rgba(107, 142, 35, .35);
}

.btn-lightBlue {
    background-color: skyblue;
    color: rgb(135, 206, 235);
    color: rgba(135, 206, 235, .35);
}

.btn-blue {
    background-color: blue;
    color: rgb(0, 0, 255);
    color: rgba(0, 0, 255, .35);
}

.btn-purple {
    background-color: purple;
    color: rgb(128, 0, 128);
    color: rgba(128, 0, 128, .35);
}

.btn-pink {
    background-color: magenta;
    color: rgb(255, 0, 255);
    color: rgba(255, 0, 255, .35);
}

#btn-user1 {
    background-color: white;
    color: rgb(255,255,255);
    color: rgba(255,255,255, .35);
}

#btn-user2 {
    background-color: white;
    color: rgb(255,255,255);
    color: rgba(255,255,255, .35);
}

#btn-user3 {
    background-color: white;
    color: rgb(255,255,255);
    color: rgba(255,255,255, .35);
}

#btn-user4 {
    background-color: white;
    color: rgb(255,255,255);
    color: rgba(255,255,255, .35);
}

.colorSelected /*Enlarge the button and put a black border around it
when it's selected as the tool/pen color*/ {
    height: 34px;
    width: 38px;
    border: 1px solid black;
}

.btnNotSelected /*Change color based on if button is selected or not*/
{
    color: #000;
    background-color: #5bc0de;
    border-color: #46b8da;
}

.btnSelected /*Change color based on if button is selected or not*/ {
    color: #fff;
    background-color: #31b0d5;
    border-color: #269abc;
}

.movedUploadedButtons {
    margin-right: 5px;
    margin-top: 5px;
    text-indent: -4px;
    color: #000;
}

.movedUploadButton {
    margin-bottom: 5px;
    font-size: 18px; /*Same as the btn-lg size in bootstrap*/
}

@media (min-width:1281px) and (max-width: 1419px) {
    .movedUploadButton {
        font-size: 16px;
    }
}

.noShow {
    width: 0;
}

html
{
    position:relative;
    min-height: 100%;
```

```
}

.footer {
    text-align: center;
    text-shadow: 1px 1px #000000;
    color: #ffffff;
    background-color: #be0f34;
    clear: both;
    margin-top: 5px;
    margin-right: 5px;
    margin-left: 5px;
    padding-right: 50px;
    overflow:hidden;
}

/*This code shows the label of a shape to the user when they click on
that shape*/
.shapeLabelToolTip /*Label popup/tooltip*/ {
    visibility: hidden;
    font-size: 14px;
    background-color: rgb(50, 50, 50);
    background-color: rgba(50, 50, 50, 0.5);
    text-shadow: 1px 1px 1px #000;
    color: #fff;
    text-align: center;
    border-radius: 6px;
    padding: 4px;
    position: absolute;
    z-index: 500;
    top: 0;
    left: 0;
}

    .shapeLabelToolTip p {
        float: left;
        border-radius: 0 10px 10px 0;
    }

#color {
    width: 40%;
    font-size: 12px;
    height: 32px;
}

#colorPickerBtn {
    margin-top: 3px;
    margin-bottom: 6px;
    padding: 6px 10px;
}

#colorwheelpicker {
    width: 100%;
    height: 100%;
    text-align: center;
}

.farbtastic {
    width: 100%;
}

/* Popup arrow */
.canvasAndImageDiv .shapeLabelToolTip::after {
    /*content: ""; */
    position: absolute;
    top: 99%;
    right: 47.74%; /*Lines the arrow up with the click location*/
    padding-top: 5px;
    border-width: 5px;
    border-style: solid;
    border-color: #555 transparent transparent transparent;
}
/***********************************************************************
*************/
#nav {
    text-align: center;
    padding-right: 15px;
    color: #ffffff;
    text-shadow: 1px 1px #000000;
    background-color: #be0f34;
    text-decoration: none;
}

#biggerFont {
    font-size: large;
}

#noPaddingLeftMarginSpace {
    padding-left: 0;
    padding-right: 0;
    margin-left: 2px;
}

#noPaddingRightMarginSpace {
    padding-left: 0;
    padding-right: 0;
    margin-right: 2px;
}

.rightMargin {
    margin-right: 2px;
}

.leftMargin {
    margin-left: 2px;
}

.hide {
    visibility: hidden;
    display: none;
}

.show {
    visibility: visible;
    display: block;
```

```
}

.moveColorsRight {
    padding-left: 12.5%;
}

#popupFormHolder {
    z-index: 900;
    width: 100%;
    height: 100%;
    opacity: .95;
    top: 0;
    left: 0;
    display: none;
    position: absolute;
    background-color: #555;
    overflow: hidden;
}
#saveSubmit {
    position: absolute;
    max-width: 75%;
    max-height: 65%;
    width: 34%;
    height: 65%;
    margin-left: 34%;
    margin-top: 10%;
    border: 3px solid gray;
    border-radius: 10px;
    background-color: rgb(255, 0, 0);
    background-color: rgba(255, 0, 0, .35);
}
#percentTesting {
    width: 75%;
    padding: 10px;
    border: 1px solid #ccc;
    font-size: 14px;
    margin-top: 5px;
}
#pixelGap {
    width: 75%;
    padding: 10px;
    border: 1px solid #ccc;
    font-size: 14px;
    margin-top:20px;
}
#formButtons {
    margin-left:11%;
    padding-left: 50px;
    padding-right: 50px;
    font-size: 16px;
    line-height: 1.5;
    border-radius: 5px;
}
#sliderValue {
    font-size: 19px;
    padding-bottom: 20px;
    line-height: 1.1;
    color: black;
    opacity: 1;
    text-align: center;
    vertical-align: middle;
}
#percentTestingValue, #pixelGapValue
{
    color: black;
    opacity: 1;
    font-size: 17px;
    margin-bottom: 15px;
    padding-top: 10px;
    text-align: center;
}
#submtInfo
{
    font-size: 12px;
    color: black;
    opacity: 1;
}

.usernamePosition
{
    position: absolute;
    top: 10px;
    right:0;
    left: 10px;
}
.logoutPosition {
    text-align: right;
    position: absolute;
    right: 10px;
    top: 10px;
    color: white;
    background-color: #be0f34;
    text-shadow: 1px 1px #000000;
    z-index: 10;
}

.InvalidCreds {
    color: #be0f34;
    text-align: center;
}
.logoutSuccess {
    color: #3c763d;
    text-align: center;
}

.centerAlign {
    text-align: center;
}
```

js

login.js

```
'use strict';
```

```
function changeText()
{
    alertify.log("Contact dhamilton@nnu.edu for help recovering your
account.");
}
function changeColor()
{
    document.getElementById("Help").style.color = '#be0f34';
}
function changeBack() {
    document.getElementById("Help").style.color = 'black';
}

function validateForm()
{
    if (document.getElementById("userPassword").value == "" ||
document.getElementById("username").value == "")
    {
        alertify.error("You must fill in all of the fields before
submitting...");
    }
    else
    {
        //Clean data before sending
        var emailTest =
document.getElementById("username").value.replace(/[^a-z,.@!#*'_0-9
]/ig, "");
        var passwordTest =
document.getElementById("userPassword").value.replace(/[^a-z,.@!#*'_0-9
]/ig, "");
        var badData = false;
        if (emailTest != document.getElementById("username").value ||
emailTest.length == 0)
        {
            badData = true;
        }
        if (passwordTest !=
document.getElementById("userPassword").value || passwordTest.length ==
0)
        {
            badData = true;
        }
        if (badData)
        {
            alertify.error("Email or password contained bad data.
Please try again.");
        }
        else
        {
            document.getElementById("LoginSubmit").submit();
        }
    }
}

function createAccountForm()
{
    location.href = "./create_account";
}
```

training_data_selector.js

```
//site.js
"use strict";
document.getElementById("color").style.visibility = "hidden"; //Hide
the color wheel input box until the image is drawn to avoid confusing
user
document.getElementById("colorPickerBtn").style.visibility = "hidden";
//Hide the color wheel button box until the image is drawn to avoid
confusing user

/************************************Size Sentive
Elements********************************************/
//Scroll Bar Text
if ($(window).height() < 600 || $(window).width() < 600) { //size for
sm-col to kick in
    document.getElementById("resizeText").innerHTML = "Slide the ruler
to resize the Image";
}
else {
    document.getElementById("resizeText").innerHTML = "Shift-Scroll to
resize Image.";
}
/**********************************************************************
*********************************/

var potraitLayout = false; //Global variable - Tells functions if the
image is portrait view or not (landscape)
var originalWidth = 0; //original image width used in ratio re-drawing
calculations
var originalHeight = 0; //original image height used in ratio re-
drawing calculations
var zoom = true; //Used in zoom scroll and resize scroll
var stage; //stage for create js (like canvas)
var bitmap; //bitmap for create js
var myGraphics = new createjs.Graphics(); //used when clearing the
stage for new drawings
var imageName; //Is the name of our file so we can output in our save
function
var labelToolTip; //Declared every time we bring in a new drawing, used
to attached our tooltip to the stage
var drawingCount = 0; //Keeps track of how many drawings are on the
screen to speed up C++ program
/*Constant values that represent my arrays for undo operations - Also
the ID of corresponding shapes*/
const POINT = 1;
const PENCIL = 2;
const LINE = 3;
const POLYLINE = 4;
const POLYGON = 5;
const CIRCLE = 6;
const AUTO_CLUSTER = 7;
const FLOOD_FILL = 8;
/**********************************************************************
*******************************/
```

```
const UNCOMPLETED_POLYGON = "uncompletedPolygon"; //Used as the name of
the line segments in an uncompleted polygon for finishing the polygon
and undo/redo/delete
const UNCOMPLETED_POLYLINE = "uncompletedPolyline"; //Used as the name
of the line segments in an uncompleted polyline for finishing the
polyline and undo/redo/delete
/******Arrays and Objects that hold our drawing data******/
//Label is what the user calls the drawing section, x and y are its
location on the bitmap, color is the color of the drawing,
var drawingData = {
        pointData: [], pencilData: [], lineData: [], polylineData: [],
polygonData: [], circleData: [], auto_clusterData: [], flood_fillData:
[],
        wipeAll: function () //function that is called when we load a
new image in.
        {
            drawingData.pointData.length = 0; //pointData holds
everything to do with point drawings
            drawingData.pencilData.length = 0; //pencilData holds
everything to do with pencil drawings
            drawingData.lineData.length = 0; //lineData holds
everything to do with pencil drawings
            drawingData.polylineData.length = 0; //polylineData holds
everything to do with polyline drawings
            drawingData.polygonData.length = 0; //polygonData holds
everything to do with polygon drawings
            drawingData.circleData.length = 0; //circleData holds
everything to do with circle drawings
            drawingData.auto_clusterData.length = 0; //auto_clusterData
holds everything to do with auto_cluster drawings
            drawingData.flood_fillData.length = 0; //flood_fillData
holds everything to do with flood_fill drawings
            drawingData.undoDrawingOrder.length = 0; //A list of all
drawings -> allows us to keep track of last drawn object (which allows
us to undo most recent)
            drawingData.undoPoly = false; //Will tell the mouse to draw
from the last polyline position (after undo)
            drawingData.shapeToDelete = null; //Holds data on the shape
we just clicked on (so we can delete it)
            drawingData.labelCount.length = 0; //Holds labels and how
many of that label ["label", labelCount, "label2", labelCount2]
            drawingData.redoDrawing.length = 0; //Holds everything that
was pushed off the undo array so we can redo if necessary
            drawingData.unusedLabels.length = 0; //An array that holds
labels that are no longer being used because the shape got deleted/undo
            drawingData.selectedObject = [0, 0]; //An array that holds
the type of drawing and which drawing in that type that it is so we can
properly select a drawing and delete it/see its label
        },
        undoDrawingOrder: [], redoDrawing: [], undoPoly: false,
shapeToDelete: null, labelCount: [], unusedLabels: [], selectedObject:
[0, 0]
    };
function pointConstr(label, x, y, color, point, pointFill) //function
that is used to create new point objects
{
    this.label = label;
    this.x = x;
    this.y = y;
    this.color = color;
    this.point = point;
    this.id = POINT;
    this.pointFill = pointFill;
}
function lineConstr(label, color, line, startX, startY, endX, endY)
//function that is used to create new line objects
{
    this.label = label;
    this.color = color;
    this.lineObject = line;
    this.x = [startX, endX];
    this.y = [startY, endY];
    this.id = LINE;
}
function pencilConstr(label, color, pencil, startX, startY) //function
that is used to create new line objects
{
    this.label = label;
    this.color = color;
    this.pencil = pencil;
    this.x = [startX];
    this.y = [startY];
    this.id = PENCIL;
}
function polylineConstr(label, color, polyline, startX, startY, endX,
endY) //function that is used to create new line objects
{
    this.label = label;
    this.color = color;
    this.polyline = polyline;
    this.x = [startX, endX];
    this.y = [startY, endY];
    this.id = POLYLINE;
}
function polygonConstr(label, color, fillColor, polygon, startX,
startY, endX, endY) //function that is used to create new line objects
{
    this.label = label;
    this.color = color;
    this.fillColor = fillColor;
    this.polygon = polygon;
    this.x = [startX, endX];
    this.y = [startY, endY];
    this.id = POLYGON;
}
function circleConstr(label, color, fillColor, circle, startX, startY,
radius) //function that is used to create new line objects
{
    this.label = label;
    this.color = color;
    this.fillColor = fillColor;
    this.circle = circle;
    this.x = startX
    this.y = startY
    this.radius = radius;
    this.id = CIRCLE;
}
/*********************************************************/

/*********Object helping determine when a polyline is completed and how
to complete a polygon*********/
var polyButtons = {
    polyFirstTime: true, reset: function () {
        this.polyFirstTime = true;
    }
};
/**********************************************************************
*******************************/

//Handles everything color based -> selecting, making look selected,
and returning color value
var color = {
    blackColor: false, whiteColor: false, redColor: false, orangeColor:
false, yellowColor: false, //Color Variables
    neonGreenColor: false, pukeGreenColor: false, lightBlueColor:
false, blueColor: false, //Color Variables
    purpleColor: false, pinkColor: false, user1Color: false,
user2Color: false, user3Color: false, user4Color: false,
currentUserColor: 1, //Color Variables
    disableColorButtons: function () //called everytime a color is
changed/picked
    {
        var ratio;
        var size = document.getElementById("size").value;
        if (potraitLayout) {
            ratio = size / originalHeight;
        }
        else {
            ratio = size / originalWidth;
        }
        polyButtons.polyFirstTime = true;
        if (drawingData.polygonData.length > 0)
        {
            if (drawingData.polygonData[drawingData.polygonData.length
- 1].polygon.name == UNCOMPLETED_POLYGON) //If we have an uncompleted
polygon on the image, complete it
            {

                drawCompletePolygon(true, ratio); //If a new color is
picked, the polygon will be completed if it hasn't been already
            }
        }
        if (drawingData.polylineData.length > 0) {
            if
(drawingData.polylineData[drawingData.polylineData.length -
1].polyline.name == UNCOMPLETED_POLYLINE) //If we have an uncompleted
polyline on the image, complete it
            {

                drawCompletePolyline(true, ratio); //If a new color is
picked, the polyline will be completed if it hasn't been already
```

```
            }
        }
        /*Sets button boolean to false which is used when determining
which color value to send to canvas - drawingColor()*/
        this.blackColor = false; this.whiteColor = false;
this.darkRedColor = false; this.redColor = false;
        this.orangeColor = false; this.yellowColor = false;
this.neonGreenColor = false; this.pukeGreenColor = false;
        this.lightBlueColor = false; this.blueColor = false;
this.purpleColor = false; this.pinkColor = false;
        this.user1Color = false; this.user2Color = false;
this.user3Color = false; this.user4Color = false;

/***********************************************************************
********************************************/

        /****Makes buttons look like they aren't selected****/
        if ($('#btn-black').hasClass('colorSelected')) {
            $('#btn-black').removeClass('colorSelected');
        }
        else if ($('#btn-white').hasClass('colorSelected')) {
            $('#btn-white').removeClass('colorSelected');
        }
        else if ($('#btn-darkRed').hasClass('colorSelected')) {
            $('#btn-darkRed').removeClass('colorSelected');
        }
        else if ($('#btn-red').hasClass('colorSelected')) {
            $('#btn-red').removeClass('colorSelected');
        }
        else if ($('#btn-orange').hasClass('colorSelected')) {
            $('#btn-orange').removeClass('colorSelected');
        }
        else if ($('#btn-yellow').hasClass('colorSelected')) {
            $('#btn-yellow').removeClass('colorSelected');
        }
        else if ($('#btn-neonGreen').hasClass('colorSelected')) {
            $('#btn-neonGreen').removeClass('colorSelected');
        }
        else if ($('#btn-pukeGreen').hasClass('colorSelected')) {
            $('#btn-pukeGreen').removeClass('colorSelected');
        }
        else if ($('#btn-lightBlue').hasClass('colorSelected')) {
            $('#btn-lightBlue').removeClass('colorSelected');
        }
        else if ($('#btn-blue').hasClass('colorSelected')) {
            $('#btn-blue').removeClass('colorSelected');
        }
        else if ($('#btn-purple').hasClass('colorSelected')) {
            $('#btn-purple').removeClass('colorSelected');
        }
        else if ($('#btn-pink').hasClass('colorSelected')) {
            $('#btn-pink').removeClass('colorSelected');
        }
        else if ($('#btn-user1').hasClass('colorSelected')) {
            $('#btn-user1').removeClass('colorSelected');
        }
        else if ($('#btn-user2').hasClass('colorSelected')) {
            $('#btn-user2').removeClass('colorSelected');
        }
        else if ($('#btn-user3').hasClass('colorSelected')) {
            $('#btn-user3').removeClass('colorSelected');
        }
        else if ($('#btn-user4').hasClass('colorSelected')) {
            $('#btn-user4').removeClass('colorSelected');
        }
        /*********************************************************/
    },
    /*********Selects button based on user input (event lister
listening to these functions************/
    black: function ()
    {
        color.disableColorButtons();
        color.blackColor = true;
        $('#btn-black').addClass('colorSelected');
    },
    white: function ()
    {
        color.disableColorButtons();
        color.whiteColor = true;
        $('#btn-white').addClass('colorSelected');
    },
    darkRed: function ()
    {
        color.disableColorButtons();
        color.darkRedColor = true;
        $('#btn-darkRed').addClass('colorSelected');
    },
    red: function ()
    {
        color.disableColorButtons();
        color.redColor = true;
        $('#btn-red').addClass('colorSelected');
    },
    orange: function ()
    {
        color.disableColorButtons();
        color.orangeColor = true;
        $('#btn-orange').addClass('colorSelected');
    },
    yellow: function ()
    {
        color.disableColorButtons();
        color.yellowColor = true;
        $('#btn-yellow').addClass('colorSelected');
    },
    neonGreen: function ()
    {
        color.disableColorButtons();
        color.neonGreenColor = true;
        $('#btn-neonGreen').addClass('colorSelected');
    },
    pukeGreen: function ()
    {
        color.disableColorButtons();
        color.pukeGreenColor = true;
        $('#btn-pukeGreen').addClass('colorSelected');
    },
    lightBlue: function()
    {
        color.disableColorButtons();
        color.lightBlueColor = true;
        $('#btn-lightBlue').addClass('colorSelected');
    },
    blue: function ()
    {
        color.disableColorButtons();
        color.blueColor = true;
        $('#btn-blue').addClass('colorSelected');
    },
    purple: function ()
    {
        color.disableColorButtons();
        color.purpleColor = true;
        $('#btn-purple').addClass('colorSelected');
    },
    pink: function()
    {
        color.disableColorButtons();
        color.pinkColor = true;
        $('#btn-pink').addClass('colorSelected');
    },
    user1: function ()
    {
        color.disableColorButtons();
        color.user1Color = true;
        $('#btn-user1').addClass('colorSelected');
    },
    user2: function ()
    {
        color.disableColorButtons();
        color.user2Color = true;
        $('#btn-user2').addClass('colorSelected');
    },
    user3: function ()
    {
        color.disableColorButtons();
        color.user3Color = true;
        $('#btn-user3').addClass('colorSelected');
    },
    user4: function ()
    {
        color.disableColorButtons();
        color.user4Color = true;
        $('#btn-user4').addClass('colorSelected');
    },

/**********************************************************************
****************************/
    drawingColor: function () //called when drawing shape -> returns
correct color value fro, which button is selected
    {
        //Color code matches the color of the button in the site.css
file
        if (color.blackColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
black")).getPropertyValue("background-color"); //returns the color
black
        }
        else if (color.whiteColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
white")).getPropertyValue("background-color"); //returns the color
white

        }
        else if (color.darkRedColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
darkRed")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.redColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
red")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.orangeColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
orange")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.yellowColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
yellow")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.neonGreenColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
neonGreen")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.pukeGreenColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
pukeGreen")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.lightBlueColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
lightBlue")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.blueColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
blue")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.purpleColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
purple")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.pinkColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
pink")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.user1Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user1")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.user2Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user2")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.user3Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user3")).getPropertyValue("background-color"); //returns the color
        }
        else if (color.user4Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user4")).getPropertyValue("background-color"); //returns the color
        }
        else //If no color has been selected, return black
        {
            return
window.getComputedStyle(document.getElementById("btn-
black")).getPropertyValue("background-color"); //returns the color
        }
    },
    fillColor: function () //Returns the fill color of the shape based
on the color button selected
    {
        //Color code matches the color of the button in the site.css
file -> color is the transparent color of background-color
        //Color does not show on the button (used for text and no text
in these buttons) so that's why I can use it here but keep the button
the correct color.
        if (color.blackColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
black")).getPropertyValue("color"); //returns the color
        }
        else if (color.whiteColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
white")).getPropertyValue("color"); //returns the color
        }
        else if (color.darkRedColor)
        {
           return window.getComputedStyle(document.getElementById("btn-
darkRed")).getPropertyValue("color"); //returns the color
        }
        else if (color.redColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
red")).getPropertyValue("color"); //returns the color
        }
        else if (color.orangeColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
orange")).getPropertyValue("color"); //returns the color
        }
        else if (color.yellowColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
yellow")).getPropertyValue("color"); //returns the color
        }
        else if (color.neonGreenColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
neonGreen")).getPropertyValue("color"); //returns the color
        }
        else if (color.pukeGreenColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
pukeGreen")).getPropertyValue("color"); //returns the color
        }
        else if (color.lightBlueColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
lightBlue")).getPropertyValue("color"); //returns the color
```

```
        }
        else if (color.blueColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
blue")).getPropertyValue("color"); //returns the color
        }
        else if (color.purpleColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
purple")).getPropertyValue("color"); //returns the color
        }
        else if (color.pinkColor)
        {
            return
window.getComputedStyle(document.getElementById("btn-
pink")).getPropertyValue("color"); //returns the color
        }
        else if (color.user1Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user1")).getPropertyValue("color"); //returns the color
        }
        else if (color.user2Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user2")).getPropertyValue("color"); //returns the color
        }
        else if (color.user3Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user3")).getPropertyValue("color"); //returns the color
        }
        else if (color.user4Color)
        {
            return
window.getComputedStyle(document.getElementById("btn-
user4")).getPropertyValue("color"); //returns the color
        }
        else //If no color has been selected, return black
        {
            return
window.getComputedStyle(document.getElementById("btn-
black")).getPropertyValue("color"); //returns the color
        }
    }
};
/****************/

/********These keep track of how many times a button has been clicked -
twice in a row means disable the button**********/
var disableDrawButton = {
    point: 0, pencil: 0, line: 0, polyline: 0, polygon: 0, circle: 0,
auto_cluster: 0, flood_fill: 0, reset: function () {
        this.point = 0; this.pencil = 0; this.line = 0; this.polyline =
0; this.polygon = 0; this.circle = 0; this.auto_cluster = 0;
        this.flood_fill = 0;
    }
};
/***********************************************************************
**************************************************/

function setVariables()
{
    if (stage.canvas.width > stage.canvas.height)
    {
        potraitLayout = false; //Global variable
    }
    else
    {
        potraitLayout = true;
    }
    alertify.set //Used for delete confirmation
        ({
        buttonReverse: true,
        labels: {
            ok: "Yes",
            cancel: "No"
        }
    });
}

$(window).resize(function () //Function that is called on window
resize. Makes it more adapt for smaller screen sizes
{
    if ($(window).height() < 600 || $(window).width() < 600) { //size
for sm-col to kick in
        document.getElementById("resizeText").innerHTML = "Slide the
ruler to resize the Image";
    }
    else
    {
        document.getElementById("resizeText").innerHTML = "Shift-Scroll
to resize Image.";
    }
    var colorWheelWidthOffset =
document.getElementById("colorpicker").offsetWidth;
    colorWheelWidthOffset = "margin-left:" + ((colorWheelWidthOffset -
195) / 2) + "px"; //195 is color wheel width
    document.getElementById("colorpicker").setAttribute("style",
colorWheelWidthOffset);
});

//Color picker upload and tutorial buttons are hidden
document.getElementById("changeUploadButtonPosition").style.visibility
= "hidden";
//Main is visible (as no picture has been selected yet)
document.getElementById("hideOriginalUploadButton").style.visibility =
"visible";
```

```
//Hide the canvas for the image until there is an image inside of it
(gets visible after it gets an image)
document.getElementById("image").style.visibility = "hidden";

/*****************The following code was adapted from
http://stackoverflow.com/questions/11406605/how-to-make-a-link-act-as-
a-file-input ********************/
//Select Image
document.getElementById("uploadButton").addEventListener("click",
selectImage);
document.getElementById("uploadButton2").addEventListener("click",
selectImage2);
document.getElementById("attributeTable-btn").addEventListener('click',
loadAttributeTableBtn); //load in attribute table
document.getElementById("tutorial").addEventListener('click',
tutorial); //tutorial button
function selectImage() //Brings up local file system to allow for an
image to be selected
{
    document.getElementById('upload').addEventListener('change',
drawImage);
    $("#upload").trigger('click');
}
function selectImage2() //Brings up local file system to allow for an
image to be selected
{
    //If the user selects a new image, then the old image will be wiped
along with everything on the stage when the new image gets loaded in
    document.getElementById('upload2').addEventListener('change',
drawImage);
    $("#upload2").trigger('click'); //Select image to upload
}
/*************************************************************************
*************************************************************************
**************/

/****Saving the drawing data was made capable by FileSaver.js****/

function submitData() {
    document.getElementById("saveSubmit").submit();
    alertify.log("Gathering your information now... In the mean time,
feel free to download the image you have created. (Click to close)",
"", 0);
    var labelCount = amountOfDifferentLabels();
    imageName = imageName.replace(/\.[^/.]+$/, "");
    var saveFileName = imageName + "_L" + labelCount + "_train";
//ImageName_L(#ofLabels)_train.csv
    //Save Image with drawings
    if (document.getElementById("downloadCanvasValue1").checked)
    {
        var myCanvas = document.getElementById("myCanvas");
        myCanvas.toBlob(function (blob) {
            saveAs(blob, saveFileName + ".jpeg");
        }, "image/jpeg");
    }
}
```

```
//popup form functions
function validateForm()
{
    if (document.getElementById("percentTesting").value == "" ||
document.getElementById("pixelGap").value == "")
    {
        alertify.error("You must fill in all of the fields before
submitting...");
    }
    else
    {
        //Clean data before sending
        var ptTest =
document.getElementById("percentTesting").value.replace(/[^0-9.]/ig,
"");
        var pgTest =
document.getElementById("pixelGap").value.replace(/[^0-9.]/ig, "");
        if (ptTest != document.getElementById("percentTesting").value
|| ptTest.length == 0)
        {
            alertify.error("Your input data for Percent Testing
contained bad data, using default value of 30% instead.");
            document.getElementById("percentTesting").value = 30;
        }
        if (pgTest != document.getElementById("pixelGap").value ||
pgTest.length == 0)
        {
            alertify.error("Your input data for Pixel Gap contained bad
data, using default value of 5 instead.");
            document.getElementById("pixelGap").value = 5;
        }
        hideForm();
    }
}
function showForm()
{
    document.getElementById("percentTesting").value = 30;
    document.getElementById("pixelGap").value = 5;
    document.getElementById("deleteDuplicatesValue1").checked = true;
    document.getElementById("downloadCanvasValue1").checked = true;
    document.getElementById("percentTestingValue").value = "30%";
    document.getElementById("pixelGapValue").value = "Pixel Gap: 5";
    document.getElementById("popupFormHolder").style.display = "block";
}
function hideForm()
{
    document.getElementById("popupFormHolder").style.display = "none";
    submitData(); //This submits the textfield we just put our
serverData into -> now Node.js has our data
}
function cancelForm()
{
    document.getElementById("popupFormHolder").style.display = "none";
}

function save(e) //Save our drawings for future use and for the
classifier
```

```
{
    e.preventDefault(); //Prevents page reload
    document.body.style.cursor = "wait"; //Shows user they need to wait
while image data is being processed
    var everythingLabeled = labelCheck(); //Make sure all drawings have
labels, if not, can't save

    if (everythingLabeled)
    {
        var serverData = ''; //Will hold everything on our array and
push it to the server. Empty so it doesn't affect anything else
        drawingCount = 0; //Reset so if they save multiple times the
count will always be accurate
        var noData = true;

        if (drawingData.pointData.length > 0 ||
drawingData.pencilData.length > 0 || drawingData.lineData.length > 0 ||
drawingData.polylineData.length > 0
         || drawingData.polygonData.length > 0 ||
drawingData.circleData.length > 0 ||
drawingData.auto_clusterData.length > 0 ||
drawingData.flood_fillData.length > 0) //if we have drawn something
        {
            noData = false;
        }
        if (noData) //If all the drawingData arrays are empty
        {
            alertify.error("There is no data to be saved. Please draw
and label some pixels first.");
        }
        else
        {
            //Save Text
            var labelCount = amountOfDifferentLabels();
            imageName = imageName.replace(/\.[^/.]+$/, "");
            var saveFileName = imageName + "_L" + labelCount +
"_train"; //ImageName_L(#ofLabels)_train.csv

            serverData = gatherServerData(serverData); //Its called
server data because at one point it was going to be just server info
but now its all of our data that will be sent to the c++ program
            drawingCount = drawingCount + "\\n"; //Convert drawingCount
into a string that has a newline character after it to match format and
make it more distinguishable from other numbers in the file
            serverData = drawingCount + serverData; //Place the drawing
count at the beginning of serverData

            document.getElementById("serverData").value = serverData;
            document.getElementById("serverDataFileName").value =
saveFileName;

            showForm();
            if (drawingData.unusedLabels.length > 0) //Have an unused
label and need to remove it from our dropdown menu
            {
                var dropDownMenu = document.getElementById("dropDown");
                for (var i = 0; i < drawingData.unusedLabels.length;
i++) //Search through unused labels
                {
                    for (var j = 0; j < dropDownMenu.length; j++)
//Search inside the dropdown menu
                    {
                        if (dropDownMenu.options[j].text ==
drawingData.unusedLabels[i]) //When match found
                        {
                            dropDownMenu.remove(j); //Remove the no
longer being used labelf
                        }
                    }
                }
            }

        }
    }
    else
    {
        alertify.error("Not all drawings are labeled. Please label your
drawings before saving.");
    }
    document.body.style.cursor = "default"; //Shows user they no longer
need to wait
}


/*****************************************************************/

/**********************************************************Load Image
into
browser***********************************************************
***********************************/
//Chrome and Firefox cannot do tif image files and IE and Edge have to
be told twice to load the image (only once though) before they actually
load it. (Not sure if still true 10/3/2016)
function drawImage(e) {
    //Check to see if browser supports file api and filereader
feautures. This will allow me to display the image to the user.
    if (window.File && window.FileReader && window.FileList &&
window.Blob) {
        /*****Code gathered from
http://stackoverflow.com/questions/6775767/how-can-i-draw-an-image-
from-the-html5-file-api-on-canvas ********/

        var URL = window.URL || window.webkitURL;
        var imageUrl, image;

        if (URL)
        {
            document.getElementById("spinner").style.visibility =
"visible";
```

```
            stage = new createjs.Stage("myCanvas");
            imageUrl = URL.createObjectURL(e.target.files[0]); //create
an url for the image so we can display it
            imageName = e.target.files[0].name; //Gets name of image to
use in our save
            clearStageForNewImage(); //Wipe stage for image to be drawn
            if (e.target.files[0].type.match('image')) //If image
upload is an image
            {
                image = new Image();
                image.src = imageUrl;
                var intlongestSide;
                image.onload = function () //function will fire after
image has loaded in
                {
                    stage.canvas.width = image.width;
                    stage.canvas.height = image.height;
                    bitmap = new createjs.Bitmap(image);
                    bitmap.name = "image";
                    stage.addChild(bitmap);
                    setVariables();
                    //User file image name may have "_" in it, which is
fine for security but need to remove so we can tell where the end of
the image name is in regards to the output csv name
                    imageName = imageName.replace(/_/g, '-');
                    intlongestSide = longestSide(image.width,
image.height);
                    setRangeOnSlider(intlongestSide);

                    //Hides the border of our canvas until the image is
drawn (its been drawn now, so making it visible)
                    document.getElementById("image").style.visibility =
"visible";
                    //Got the idea for these two lines of code below
from
                    //http://stackoverflow.com/questions/6497373/make-
content-horizontally-scroll-inside-a-div
                    // and
http://stackoverflow.com/questions/3437786/get-the-size-of-the-screen-
current-web-page-and-browser-window
                    var windowHeight = $(window).height();
                    $('#scaledImage').css("height", (windowHeight -
50));
                    document.getElementById("footer").style.position =
"relative";
                    //Move Tutorial and Select training image button
below color picker so theyre out of the way

document.getElementById("changeUploadButtonPosition").style.visibility
= "visible";

document.getElementById("hideOriginalUploadButton").style.visibility =
"hidden";
                    //Bump our canvas up to match the Y location of
other divs
document.getElementById("hideOriginalUploadButton").style.marginTop =
"-125px";
                    labelToolTip = new createjs.Text("Hello World",
"14px Arial", "#ffffff");
                    labelToolTip.shadow = new
createjs.Shadow("#000000", 2, 2, 7);
                    originalHeight = image.height;
                    originalWidth = image.width;
                    grabDimensions(image.width, image.height);
                    hasDrawn();
                    stage.update();
                    document.getElementById("spinner").style.visibility
= "hidden";
                    color.black(); //Set to black on load so that the
user knows black is the color being selected
                    /********Color Wheel Code********/
                    $('#colorpicker').farbtastic('#color');
//colorwheel
                    document.getElementById("color").style.visibility =
"visible";

document.getElementById("colorPickerBtn").style.visibility = "visible";
                    var colorWheelWidthOffset =
document.getElementById("colorpicker").offsetWidth;
                    colorWheelWidthOffset = "margin-left:" +
((colorWheelWidthOffset - 195) / 2) + "px"; //195 is colorwheel width

document.getElementById("colorpicker").setAttribute("style",
colorWheelWidthOffset);
                    /********************************/
                };
            }
            else //Not an image file
            {
                switchAlertLabelForAlert();
                alertify.alert("The file selected was not an image
file. Please select an image to train on such as a JPEG, PNG, GIF,
TIFF, or BMP.");
                alertify.error("Image upload cancelled.");
                switchAlertLabelForConfirm();
                document.getElementById("spinner").style.visibility =
"hidden"; //Turn off from turning on before if statement
            }

        }
        else
        {
            switchAlertLabelForAlert();
            alertify.alert("There was an error in rendering your image,
please reload the page or try again in a different browser.");
            switchAlertLabelForConfirm();
        }

/**********************************************************************
*******************************************************************/
    }
    else
    {
        switchAlertLabelForAlert();
        alertify.alert("We're sorry, but the File APIs are not fully
supported in this browser Please try a different browser.");
        switchAlertLabelForConfirm();
    }

/*********************************************************************
**********************************************************************
****************************************/
}
/*********************************************************************
**********************************************************************
***************************************/

/**************Grab dimensions to properly draw the canvas
function*************/
function grabDimensions(canvasWidth, canvasHeight)
{
    var dimensions = "(" + canvasWidth + "," + canvasHeight + ")";
    document.getElementById('dimensions').setAttribute("value",
dimensions);
}
/*********************************************************************
*********/

/*************************Sets maxium range and default value of range
slider******************************/
function setRangeOnSlider(longestSide)
{
    var size = document.getElementById("size");
    document.getElementById("size").setAttribute("max",
(longestSide*2));
    document.getElementById("size").setAttribute("defaultValue",
longestSide);
    document.getElementById("size").setAttribute("value", longestSide);
    size.value = longestSide;
}
/*********************************************************************
*************************************/

/*************************Return longest slide for scaling
purposes***************************/
function longestSide(width, height)
{
    if (height > width)
    {
        potraitLayout = true;
        return height;

    }
    else
    {
        potraitLayout = false;
        return width;
    }
```

```
}
/**********************************************************************
************************/

/********Code for resizing image in canvas*************/
function resize()
{
    document.getElementById("spinner").style.visibility = "visible";
    var size = document.getElementById("size").value;
    stage.removeAllChildren(); //Clear everything
    stage.addChild(bitmap); //Add our image back on
    if (potraitLayout == false) //if width is longer than height
    {
        var ratio = size / originalWidth;
        bitmap.scaleX = ratio;
        bitmap.scaleY = ratio;
        stage.canvas.width = originalWidth * ratio;
        stage.canvas.height = originalHeight * ratio;
        resizeRedraw(ratio); //Redraws any shapes in their correct
location
    }
    else //Height is longer than width
    {
        var ratio = size / originalHeight;
        bitmap.scaleX = ratio;
        bitmap.scaleY = ratio;
        stage.canvas.width = originalWidth * ratio;
        stage.canvas.height = originalHeight * ratio;
        resizeRedraw(ratio); //Redraws any shapes in their correct
location
    }
    stage.scaleX = 1; //Reset the scale (essentially resets
"zoomResize()")
    stage.scaleY = 1; //Resey the scale (essentially resets
"zoomResize()")
    keepImageInsideStageBoundaries(); //Keeps the image inside the
stage -> properly sets stage.reg and other stage properties so that the
stage will line up with the image)
    grabDimensions(stage.canvas.width, stage.canvas.height); //Update
Dimensions
    stage.update();
    document.getElementById("spinner").style.visibility = "hidden";
}
/************************************************************/

/*********Launch my event listeners as the image is drawn for the first
time*************/
function hasDrawn() //Called after image loaded so that the event
listener will be put into action.
{
    document.getElementById("myCanvas").addEventListener("mouseover",
displayCoordinates); //Displays mouse coordinates when over canvas
    document.getElementById("myCanvas").addEventListener("mouseover",
scrollResize); //Scales the image with by scrolling
    document.getElementById("myCanvas").addEventListener("mouseover",
zoomResize); //Zooms in and out the image by scrolling
```

```
    document.getElementById("myCanvas").addEventListener("mouseover",
zoomDrag); //Allows the user to move the image around after its been
zoomed in on
    stage.addEventListener("click", grabShapeObject); //Grabs any shape
the user has drawn on the stage
    document.getElementById("size").addEventListener("click", resize);
//If you click on the slider it will resize where you clicked too
    document.getElementById("scaleOut").addEventListener("click",
scaleOutButton); //Scale out button
    document.getElementById("scaleIn").addEventListener("click",
scaleInButton); //Scale in button
    document.getElementById("pointBtn").addEventListener("click",
drawPoint); //draw point
    document.getElementById("pencilBtn").addEventListener("click",
drawPencil); //draw pencil
    document.getElementById("lineBtn").addEventListener("click",
drawLine); //draw line
    document.getElementById("polylineBtn").addEventListener("click",
drawPolyline); //draw polyline
    document.getElementById("polygonBtn").addEventListener("click",
drawPolygon); //draw polygon
    document.getElementById("circleBtn").addEventListener("click",
drawCircle); //draw circle

document.getElementById("auto_clusterBtn").addEventListener("click",
drawAuto_Cluster); //draw draw auto-cluster
    document.getElementById("flood_fillBtn").addEventListener("click",
drawFlood_Fill); //draw flood_fill
    document.getElementById("save").addEventListener('click', save);
//Save
    document.getElementById("undo").addEventListener('click', undo);
//Undo
    document.getElementById("redo").addEventListener('click', redo);
//Redo
    document.getElementById("delete-btn").addEventListener('click',
deleteDrawing); //delete drawing
    document.getElementById("load-btn").addEventListener('click',
loadDrawingButton); //load in a previous drawing to make edits
    document.getElementById("btnLabel").addEventListener("click",
label); //Label button
    document.getElementById("labelForm").addEventListener("submit",
label); //listens for enter key
    document.getElementById("tutorial2").addEventListener('click',
tutorial); //tutorial for app
    document.getElementById("attributeTable-
btn2").addEventListener('click', loadAttributeTableBtn); //load in
attribute table
    document.getElementById("dropDown").addEventListener("change",
displayShapeByLabel); //drop down menu -> drawing only labels selected
    /**********************************Color
Buttons***************************************/
    document.getElementById("btn-black").addEventListener("click",
color.black); //Black
    document.getElementById("btn-white").addEventListener("click",
color.white); //White
    document.getElementById("btn-darkRed").addEventListener("click",
color.darkRed); //Dark Red
```

```
    document.getElementById("btn-red").addEventListener("click",
color.red); //Red
    document.getElementById("btn-orange").addEventListener("click",
color.orange); //Orange
    document.getElementById("btn-yellow").addEventListener("click",
color.yellow); //Yellow
    document.getElementById("btn-neonGreen").addEventListener("click",
color.neonGreen); //Neon Green
    document.getElementById("btn-pukeGreen").addEventListener("click",
color.pukeGreen); //Puke Green
    document.getElementById("btn-lightBlue").addEventListener("click",
color.lightBlue); //Light Blue
    document.getElementById("btn-blue").addEventListener("click",
color.blue); //Blue
    document.getElementById("btn-purple").addEventListener("click",
color.purple); //Purple
    document.getElementById("btn-pink").addEventListener("click",
color.pink); //Pink
    document.getElementById("btn-user1").addEventListener("click",
color.user1); //User1 (User ones are colors that the user can pick from
a color wheel (not implemented yet)
    document.getElementById("btn-user2").addEventListener("click",
color.user2); //User2
    document.getElementById("btn-user3").addEventListener("click",
color.user3); //User3
    document.getElementById("btn-user4").addEventListener("click",
color.user4); //User4
    document.getElementById("colorPickerBtn").addEventListener("click",
assignColor); //color wheel button assigns color to a color.user button

document.getElementById("colorWheelForm").addEventListener("submit",
colorWheelText); //listens for enter key

/**********************************************************************
****************/

}
/**********************************************************************
**/

/************This function uses modified code from
http://stackoverflow.com/questions/7154967/how-to-detect-scroll-
direction/33334461#33334461 ********/
function scrollResize() //Tells what direction the mouse was scrolled
and sets the range/slider bar appropriately
{
    //Keydown code taken from
http://stackoverflow.com/questions/3149362/capture-key-press-or-
keydown-event-on-div-element
    $("#scaledImage").off("keydown"); //Prevents multiple keydown
events from being triggered.
    $('#scaledImage').on('keydown', function (event)
    {
        if (event.which == 16) //16 is the value for the shift key
        {
            zoom = false;
            if (zoom == false)
            {
                var size = document.getElementById("size");
                $("#scaledImage").off('wheel'); //Prevents multiple
scroll events from being triggered
                $("#scaledImage").on('wheel', function (e) {
                    stage.scaleX = 1.1; //Reset zoom in / zoom out
                    stage.scaleY = 1.1; //Reset zoom in / zoom out
                    var delta = e.originalEvent.deltaY;
                    if (delta > 0) {
                        var intSize = parseInt(size.value, 10);
                        if (intSize > 1000) //Changes the scale at
which we resize
                        {
                            intSize = intSize - 100;
                            size.step = "100";
                        }
                        else {
                            intSize = intSize - 10;
                            size.step = "10";
                        }

                        var stringSize = intSize.toString();

document.getElementById("size").setAttribute("value", stringSize);
                        size.value = stringSize;
                    }
                    else {
                        var intSize = parseInt(size.value, 10);
                        if (intSize >= 1000) //Changes the scale at
which we resize
                        {
                            intSize = intSize + 100;
                            size.step = "100";
                            var intStep = parseInt(size.step, 10);
                            if (intSize >= size.max) //Makes the image
exactly twice as big as the original.
                            {
                                //Calculates the step amount from
current location to max location then sets that step amount.
                                var intMax = parseInt(size.max, 10);
                                intStep = intSize - intMax;
                                intStep = 100 - intStep; //normalize
intStep
                                var stringStep = intStep.toString();
                                size.step = stringStep;

                                intSize = intMax; //Set size
                            }
                        }
                        else {
                            intSize = intSize + 10;
                            size.step = "10";
                        }
                        var stringSize = intSize.toString();
                        size.value = stringSize;
                    }

                    resize();

                });

            }
            else {
                return;
            }
        }
        else {
            zoom = true;
            return;
        }
    });

    $('#scaledImage').on('keyup', function (event) {
        zoom = true;
        $("#scaledImage").off('wheel'); //Prevent resizing after key up
    });

    $(function () {
        $('#scaledImage').focus();
    });
}
/***********************************************************************
***********************************************************************
***********/

/**********Shows my coordinates**********/
function displayCoordinates()
{
    //Coordinate finder - Code heavily modified from
http://stackoverflow.com/questions/32360656/get-coordinates-of-clicked-
on-image-with-jquery
    $('#myCanvas').mousemove(function (event) {
        var local = stage.globalToLocal(stage.mouseX, stage.mouseY);
        local.x = local.x | 0; //Removes decimals
        local.y = local.y | 0; //Removes decimals
        var currentCoordinates = local.x + ', ' + local.y;
        $('#coordinates').html(window.current_coords);
        var coordinates = "(" + local.x + "," + local.y + ")";
        document.getElementById('coordinates').setAttribute("value",
coordinates);
    });
}
/****************************************/

/**********This function uses modified code from
http://jsfiddle.net/kz0dL78k/ **********/
function zoomResize()
{
    $("#myCanvas").off("wheel"); //Prevents multiple scroll events from
being triggered
    $("#myCanvas").on('wheel', function (e)
    {
        e.preventDefault(); //Prevent page from scrolling
        if (zoom) //If we aren't going the resize scroll
        {
            var delta = e.originalEvent.deltaY;
            var zoomRatio;
            var local = stage.globalToLocal(stage.mouseX,
stage.mouseY);
            if (delta < 0) //Zooming in
            {
                zoomRatio = 1.1;
                stage.regX = local.x; //View port
                stage.regY = local.y; //View port
                stage.x = stage.mouseX; //Mouse location on stage
                stage.y = stage.mouseY; //Mouse location on stage
                stage.scaleX = stage.scaleY *= zoomRatio;
            }
            else //Zooming out
            {
                zoomRatio = 1 / 1.1;
                if (stage.scaleX * zoomRatio <= 1) {
                    zoomRatio = 1;
                    return; //Saves time because we don't need to check
anything else
                }
                stage.scaleX = stage.scaleY *= zoomRatio;
                /*Set where the stage is and the viewport based on the
global mouse coordinates and the local mouse coordinates*/
                stage.regX = local.x; //View port
                stage.regY = local.y; //View port
                stage.x = stage.mouseX; //Mouse location on stage
                stage.y = stage.mouseY; //Mouse locaton on stage

/***********************************************************************
*****************************************/
                keepImageInsideStageBoundaries();

            }
            stage.update();
        }
    });
}
/***********************************************************************
*******************/

/***********This function uses modified code from
http://jsfiddle.net/kz0dL78k/ **********/
function zoomDrag()
{
    stage.removeAllEventListeners("stagemousedown"); //Prevent event
from triggering twice
    stage.addEventListener("stagemousedown", function (e) {
        var drawing = currentlyDrawing();
        if (!drawing) //If aren't drawing, then we can drag. If we are,
we have to unselect the tool to keep dragging
        {
            var offset = { x: stage.x - e.stageX, y: stage.y - e.stageY
};
```

```
            stage.removeAllEventListeners("stagemousemove");
            stage.addEventListener("stagemousemove", function (ev)
            {
                var previousStage = { x: stage.x, y: stage.y };
                stage.x = ev.stageX + offset.x; //Update Stage
                stage.y = ev.stageY + offset.y; //Update Stage

                var localWalls =
                    {
                        topLeftCorner: stage.globalToLocal(0, 0),
                        bottomRightCorner:
stage.globalToLocal(stage.canvas.width, stage.canvas.height)
                    }; //Location of local walls (the boundries that
you see, zoomed in or not)

                if (localWalls.topLeftCorner.x <= 0) //If overflow on
the right side (backwards I know)
                {
                    stage.x = previousStage.x; //Out of bounds, reset
stage to last location
                }
                else if (localWalls.bottomRightCorner.x >=
stage.canvas.width) //If overflow on the left side (backwards I know)
                {
                    stage.x = previousStage.x //Out of bounds, reset
stage to last location
                }
                else {
                    //Keep update
                }

                if (localWalls.topLeftCorner.y <= 0) //If overflow on
the bottom (backwards I know)
                {
                    stage.y = previousStage.y; //Out of bounds, reset
stage to last location
                }
                else if (localWalls.bottomRightCorner.y >=
stage.canvas.height) //If overflow on the top (backwards I know)
                {
                    stage.y = previousStage.y; //Out of bounds, reset
stage to last location
                }
                else {
                    //Keep update
                }
                stage.update();
            });
            stage.removeAllEventListeners("stagemouseup"); //Prevent
event from triggering twice
            stage.addEventListener("stagemouseup", function () {
                stage.removeAllEventListeners("stagemousemove");
            });
        }
    });
}
```

```
/***********************************************************************
*******************/

/*************Scale out and in buttons next to slider************/
function scaleOutButton()
{
    stage.scaleX = 1.1; //Reset zoom in / zoom out
    stage.scaleY = 1.1; //Reset zoom in / zoom out
    var size = document.getElementById("size");
    var intSize = parseInt(size.value, 10);
    if (intSize > 1000) //Changes the scale at which we resize
    {
        intSize = intSize - 100;
        size.step = "100";
    }
    else {
        intSize = intSize - 10;
        size.step = "10";
    }

    var stringSize = intSize.toString();
    document.getElementById("size").setAttribute("value", stringSize);
    size.value = stringSize;

    resize(); //Redraw image now
}
function scaleInButton()
{
    stage.scaleX = 1.1; //Reset zoom in / zoom out
    stage.scaleY = 1.1; //Reset zoom in / zoom out
    var size = document.getElementById("size");
    var intSize = parseInt(size.value, 10);
    if (intSize >= 1000) //Changes the scale at which we resize
    {
        intSize = intSize + 100;
        size.step = "100";
        var intStep = parseInt(size.step, 10);
        if (intSize >= size.max) //Makes the image exactly twice as big
as the original.
        {
            //Calculates the step amount from current location to max
location then sets that step amount.
            var intMax = parseInt(size.max, 10);
            intStep = intSize - intMax;
            intStep = 100 - intStep; //normalize intStep
            var stringStep = intStep.toString();
            size.step = stringStep;

            intSize = intMax; //Set size
        }
    }
    else {
        intSize = intSize + 10;
        size.step = "10";
    }
    var stringSize = intSize.toString();
    size.value = stringSize;
    resize(); //Redraw image now
}
/*****************************************************************/

/*These functions are where all our drawing magic happens*/
function drawPoint()
{
    var disableValue = disableDrawButton.point; //Temp variable that
will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.point = disableValue; //place temp value back in
    disableDrawButton.point++;
    if (disableDrawButton.point >= 2) //Reset counter check
    {
        disableDrawButton.point = 0;
    }
    if (disableDrawButton.point >= 1) //Checks to see if this is our
second click or not
    {
        $('#pointBtn').addClass('btnSelected'); //Shows the user that
this button is currently selected
        $('#pointBtn').removeClass('btnNotSelected'); //Shows the user
that this button is currently selected
        setCursorForDrawing();
        swapPolyButtons(3); //Remove poly buttons
        //Draws on the canvas
        $('#myCanvas').click(function ()
        {
            if (currentLabelBeingShown())//Checks to see if this is our
second click or not
            {
                /*****************************Set characteristics of
shape*****************************************/
                var point = new createjs.Shape();
                stage.addChild(point);
                point.name = Date.now();
                point.id = POINT;
                point.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
                var drawingColor = color.drawingColor();
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                var local = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                point.graphics.setStrokeStyle(.25); //Line width
                point.graphics.beginStroke(drawingColor);
                point.graphics.beginFill(drawingColor);
                var pointFill =
point.graphics.beginFill(drawingColor).command;
                point.graphics.drawRect(local.x, local.y, 1, 1);
//Places 1 pixel dot at location of mouse
/***********************************************************************
*************************/
                //Create shape and then add to the stage
                stage.update();
                if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                {
                    var ratio;
                    if (potraitLayout) {
                        ratio = originalHeight / stage.canvas.height;
                    }
                    else {
                        ratio = originalWidth / stage.canvas.width;
                    }
                    local.x = local.x * ratio; //Update local x for
array storage
                    local.y = local.y * ratio; //Update local y for
array storage
                }
                var newPoint = new pointConstr(null, local.x, local.y,
drawingColor, point, pointFill); //label is null because the user has
yet to set it
                drawingData.pointData.push(newPoint); //Push object
onto the point array
                drawingData.undoDrawingOrder.push(POINT); //Pushing a
POINT means the pointTool was used, so if we undo any drawings, we undo
from the pointData array
                drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
            }
        });
    }
}
function drawPencil() {
    var disableValue = disableDrawButton.pencil; //Temp variable that
will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.pencil = disableValue; //place temp value back in
    disableDrawButton.pencil++;
    if (disableDrawButton.pencil >= 2) //Reset counter check
    {
        disableDrawButton.pencil = 0;
    }

    if (disableDrawButton.pencil >= 1)//Checks to see if this is our
second click or not
    {

        $('#pencilBtn').addClass('btnSelected'); //Shows the user that
this button is currently selected
        $('#pencilBtn').removeClass('btnNotSelected'); //Shows the user
that this button is currently selected
        swapPolyButtons(3); //Remove poly buttons
        setCursorForDrawing();
        var pencil;
        var move = false;
        var startCoordinates = { x: 0, y: 0 };
        var endCoordinates = { x: 0, y: 0 };

        function handleMouseDown() {
            startCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY);
            pencil = new createjs.Shape();
            stage.addChild(pencil);
            pencil.name = Date.now();
            pencil.id = PENCIL;
            pencil.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            pencil.graphics.setStrokeStyle(1); //Line width
            var drawingColor = color.drawingColor();
            drawingColor = rgb2hex(drawingColor); //Convert rgb to hex
and store color as a hex value
            pencil.graphics.beginStroke(drawingColor);
            pencil.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
            move = true;
            if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
            {
                var ratio;
                if (potraitLayout) {
                    ratio = originalHeight / stage.canvas.height;
                }
                else {
                    ratio = originalWidth / stage.canvas.width;
                }
                startCoordinates.x = startCoordinates.x * ratio;
//Update start x for array storage
                startCoordinates.y = startCoordinates.y * ratio;
//Update start y for array storage
            }
            var newPencil = new pencilConstr(null, drawingColor,
pencil, startCoordinates.x, startCoordinates.y); //label is null
because the user has yet to set it, pass the line object so I can
remove it later
            drawingData.undoDrawingOrder.push(PENCIL); //Pushing a LINE
means the lineTool was used, so if we undo any drawings, we undo from
the lineData array
            drawingData.redoDrawing.length = 0; //Empty my redo array
as I just added something - Will need to do this everytime I draw
something
            drawingData.pencilData.push(newPencil);
        }

        function handleMouseMove() {
            if (move) {
                var local = stage.globalToLocal(stage.mouseX,
stage.mouseY);
```

```
                pencil.graphics.lineTo(local.x, local.y);
                if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                {
                    var ratio;
                    if (potraitLayout) {
                        ratio = originalHeight / stage.canvas.height;
                    }
                    else {
                        ratio = originalWidth / stage.canvas.width;
                    }
                    local.x = local.x * ratio; //Update end x for array
storage
                    local.y = local.y * ratio; //Update end y for array
storage
                }
                if
(drawingData.pencilData[drawingData.pencilData.length -
1].x[drawingData.pencilData[drawingData.pencilData.length - 1].x.length
- 1] == local.x &&

drawingData.pencilData[drawingData.pencilData.length -
1].y[drawingData.pencilData[drawingData.pencilData.length - 1].y.length
- 1] == local.y) {
                    //Don't push as these are the same points as the
previous points
                }
                else //New points
                {

drawingData.pencilData[drawingData.pencilData.length -
1].x.push(local.x);

drawingData.pencilData[drawingData.pencilData.length -
1].y.push(local.y);
                }
                stage.update();
            }
        }

        function handleMouseUp() {
            if (move) {
                endCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY);
                pencil.graphics.lineTo(endCoordinates.x,
endCoordinates.y);
                pencil.graphics.endStroke;
                stage.update();
                move = false;
                if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                {
                    var ratio;
                    if (potraitLayout) {
                        ratio = originalHeight / stage.canvas.height;
                    }
                    else {
                        ratio = originalWidth / stage.canvas.width;
                    }
                    endCoordinates.x = endCoordinates.x * ratio;
//Update end x for array storage
                    endCoordinates.y = endCoordinates.y * ratio;
//Update end y for array storage
                }
                if
(drawingData.pencilData[drawingData.pencilData.length -
1].x[drawingData.pencilData[drawingData.pencilData.length - 1].x.length
- 1] == endCoordinates.x &&

drawingData.pencilData[drawingData.pencilData.length -
1].y[drawingData.pencilData[drawingData.pencilData.length - 1].y.length
- 1] == endCoordinates.y) {
                    //Don't push as these are the same points as the
previous points
                }
                else //New points
                {

drawingData.pencilData[drawingData.pencilData.length -
1].x.push(endCoordinates.x);

drawingData.pencilData[drawingData.pencilData.length -
1].y.push(endCoordinates.y);
                }
            }
        }
        $("#myCanvas").mousedown(function ()
        {
            if (currentLabelBeingShown())//Checks to see if this is our
second click or not
            {
                handleMouseDown();
            }
            else
            {
                move = false; //If false, prevent mousemove, mouseup,
and mouseout from firing.
            }
        });
        $("#myCanvas").mousemove(function () { handleMouseMove(); });
        $("#myCanvas").mouseup(function () { handleMouseUp(); });
        $("#myCanvas").mouseout(function () { handleMouseUp(); });
    }
}
function drawLine()
{
    var disableValue = disableDrawButton.line; //Temp variable that
will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.line = disableValue; //place temp value back in
    disableDrawButton.line++;
    if (disableDrawButton.line >= 2) //Reset counter check
    {
        disableDrawButton.line = 0;
    }
    if (disableDrawButton.line >= 1)//Checks to see if this is our
second click or not
    {
        $('#lineBtn').addClass('btnSelected'); //Shows the user that
this button is currently selected
        $('#lineBtn').removeClass('btnNotSelected'); //Shows the user
that this button is currently selected
            swapPolyButtons(3); //Remove poly buttons
            setCursorForDrawing();
            var startCoordinates = { x: 0, y: 0 }; //Will change based
upon where the user clicked
            var move, outOfBounds = false;
            var line;
            function handleMouseDown() {
                line = new createjs.Shape();
                line.name = Date.now();
                line.id = LINE;
                line.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
                stage.addChild(line);
                startCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                move = true;
            }
            function handleMouseMove() {
                if (move) {
                    line.graphics.clear(); //Wipe so old drawing won't
show
                    var drawingColor = color.drawingColor();
                    var local = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                    drawingColor = rgb2hex(drawingColor); //Convert rgb
to hex and store color as a hex value
                    line.graphics.setStrokeStyle(1); //Line width
                    line.graphics.beginStroke(drawingColor);
                    line.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
                    line.graphics.lineTo(local.x, local.y);
                    line.graphics.endStroke();
                    stage.update();
                }
            }
            function handleMouseUp() {
                if (move) {
                    //Draw final line
                    line.graphics.clear(); //Wipe so old drawings don't
show
                    stage.update(); //Update the clear so the user
doesn't grab pixels they've colored
                    var endCoordinates =
stage.globalToLocal(stage.mouseX, stage.mouseY); //Mouse location in
local coordinates
```

```
                    var drawingColor = color.drawingColor();
                    drawingColor = rgb2hex(drawingColor); //Convert rgb
to hex and store color as a hex value
                    line.graphics.setStrokeStyle(1); //Line width
                    line.graphics.beginStroke(drawingColor);
                    line.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
                    line.graphics.lineTo(endCoordinates.x,
endCoordinates.y);
                    line.graphics.endStroke();

                    move = false;

                    //Done drawing, push data to our arrays now

                    stage.update();
                    if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                    {
                        var ratio;
                        if (potraitLayout) {
                            ratio = originalHeight /
stage.canvas.height;
                        }
                        else {
                            ratio = originalWidth / stage.canvas.width;
                        }
                        startCoordinates.x = startCoordinates.x *
ratio; //Update start x for array storage
                        startCoordinates.y = startCoordinates.y *
ratio; //Update start y for array storage
                        endCoordinates.x = endCoordinates.x * ratio;
//Update end x for array storage
                        endCoordinates.y = endCoordinates.y * ratio;
//Update end y for array storage
                    }
                    var newLine = new lineConstr(null, drawingColor,
line, startCoordinates.x, startCoordinates.y, endCoordinates.x,
endCoordinates.y); //label is null because the user has yet to set it,
pass the line object so I can remove it later
                    drawingData.undoDrawingOrder.push(LINE); //Pushing
a LINE means the lineTool was used, so if we undo any drawings, we undo
from the lineData array
                    drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
                    drawingData.lineData.push(newLine);
                }
            }

            $("#myCanvas").mousedown(function ()
            {
                if (currentLabelBeingShown())//Checks to see if this is
our second click or not
                {
                    handleMouseDown();
                }
                else
                {
                    move = false;
                }
            });
            $("#myCanvas").mousemove(function () { handleMouseMove();
});
            $("#myCanvas").mouseup(function () { handleMouseUp(); });
            $("#myCanvas").mouseout(function () { handleMouseUp(); });
    }
}
function drawPolyline() {
    var disableValue = disableDrawButton.polyline; //Temp variable that
will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.polyline = disableValue; //place temp value back
in
    disableDrawButton.polyline++;
    if (disableDrawButton.polyline >= 2) //Reset counter check
    {
        disableDrawButton.polyline = 0;
    }

    if (disableDrawButton.polyline >= 1)//Checks to see if this is our
second click or not
    {
        $('#polylineBtn').addClass('btnSelected'); //Shows the user
that this button is currently selected
        $('#polylineBtn').removeClass('btnNotSelected'); //Shows the
user that this button is currently selected
        setCursorForDrawing();
        swapPolyButtons(2); //Adds polyline confirm button
        var startCoordinates = { x: 0, y: 0 }; //Will change based upon
where the user clicked
        var beginningCoordinates = { x: 0, y: 0 }; //Used for the start
point of the polyline (not line segments)
        var start = { x: 0, y: 0 };//Used to place polyline start point
on old end point
        var move, outOfBounds = false;
        polyButtons.polyFirstTime = true;
        var polyline;
        var endCoordinates; //Used in two functions, mouse up and
polyline confirmation

        function handleMouseDown() {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = size / originalHeight;
            }
            else {
                ratio = size / originalWidth;
            }
            polyline = new createjs.Shape();
            polyline.name = UNCOMPLETED_POLYLINE; //Name will become
more specific once the polygon is completed ->This is the name of only
one side, which we will be deleting when we form the whole polygon
            polyline.id = POLYLINE;
            polyline.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            stage.addChild(polyline);
            if (polyButtons.polyFirstTime) //If this is a new polygon
            {
                startCoordinates =
stage.globalToLocal(stage.mouseX,stage.mouseY); //Mouse location in
local coordinates
                beginningCoordinates = { x: startCoordinates.x, y:
startCoordinates.y }; //Round the coordinates down for recording so
user can close the polygon easier
            }
            else //Else we are continuing one we've already started
            {
                if (drawingData.redoDrawing.length == 0) {
                    drawingData.undoPoly = false;
                }
                if (drawingData.undoPoly) {

                    startCoordinates.x =
(drawingData.polylineData[drawingData.polylineData.length -
1].x[drawingData.polylineData[drawingData.polylineData.length -
1].x.length - 1] * ratio);
                    startCoordinates.y =
(drawingData.polylineData[drawingData.polylineData.length -
1].y[drawingData.polylineData[drawingData.polylineData.length -
1].y.length - 1] * ratio);
                }
                else {
                    startCoordinates.x = start.x;
                    startCoordinates.y = start.y;
                }

            }
            move = true;
        }
        function handleMouseMove() {
            if (move) {
                polyline.graphics.clear(); //Wipe so old drawing won't
show
                var drawingColor = color.drawingColor();
                var local = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                polyline.graphics.setStrokeStyle(1); //Line width
                polyline.graphics.beginStroke(drawingColor);
                polyline.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
                polyline.graphics.lineTo(local.x, local.y);
                polyline.graphics.endStroke();
                stage.update();
            }
        }
        function handleMouseUp() {
            if (move) {
                move = false;
                //Draw final line
                polyline.graphics.clear(); //Wipe so old drawings don't
show
                stage.update(); //Update the clear so the user doesn't
grab pixels they've colored
                endCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                var drawingColor = color.drawingColor();
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                polyline.graphics.setStrokeStyle(1); //Line width
                polyline.graphics.beginStroke(drawingColor);
                polyline.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
                polyline.graphics.lineTo(endCoordinates.x,
endCoordinates.y);
                polyline.graphics.endStroke();

                stage.update();
                //Set the end points of this line to the start points
of the next line
                start.x = endCoordinates.x;
                start.y = endCoordinates.y;
                if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                {
                    var ratio;
                    if (potraitLayout) {
                        ratio = originalHeight / stage.canvas.height;
                    }
                    else {
                        ratio = originalWidth / stage.canvas.width;
                    }
                    if (polyButtons.polyFirstTime) //Only need to do
the beginning for the first line, then its just the end coordinates
that get pushed to our array
                    {
                        beginningCoordinates.x = beginningCoordinates.x
* ratio; //Update start x for array storage
                        beginningCoordinates.y = beginningCoordinates.y
* ratio; //Update start y for array storage
                    }
                    endCoordinates.x = endCoordinates.x * ratio;
                    endCoordinates.y = endCoordinates.y * ratio;
                }
                if (polyButtons.polyFirstTime)
                {
                    var newPolyline = new polylineConstr(null,
drawingColor, polyline, beginningCoordinates.x, beginningCoordinates.y,
endCoordinates.x, endCoordinates.y); //label is null because the user
has yet to set it, pass the line object so I can remove it later
```

```
                    drawingData.undoDrawingOrder.push(POLYLINE);
//Pushing a POLYLINE means the polylineTool was used, so if we undo any
drawings, we undo from the polylineData array
                    drawingData.polylineData.push(newPolyline);
                    drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
                }
                else
                {

drawingData.polylineData[drawingData.polylineData.length -
1].x.push(endCoordinates.x); //Push the end points of the line to the
end of our polyline data

drawingData.polylineData[drawingData.polylineData.length -1
].y.push(endCoordinates.y); //Push the end points of the line to the
end of our polyline data
                    drawingData.undoDrawingOrder.push(POLYLINE);
                    drawingData.redoDrawing.length = 0;
                }
                polyButtons.polyFirstTime = false; //No longer making
the begining of a polygon
            }
        }
        function polylineConfirmation()
        {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = stage.canvas.height / originalHeight;
            }
            else {
                ratio = stage.canvas.width / originalWidth;
            }
            polyButtons.polyFirstTime = true; //No longer making the
begining of a polygon
            drawCompletePolyline(true, ratio); // true means we are
only completing the last polygon drawn (this one only and not all of
them)
        }

        $("#myCanvas").mousedown(function ()
        {
            if (currentLabelBeingShown())//Checks to see if this is our
second click or not
            {
                handleMouseDown();
            }
            else
            {
                move = false;
            }
        });
        $("#myCanvas").mousemove(function () { handleMouseMove(); });
        $("#myCanvas").mouseup(function () { handleMouseUp(); });
        $("#myCanvas").mouseout(function () { handleMouseUp(); });
        $("#polylineConfirmation").click(function () {
polylineConfirmation(); });

    }
}
function drawPolygon()
{
    var disableValue = disableDrawButton.polygon; //Temp variable that
will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.polygon = disableValue; //place temp value back
in
    disableDrawButton.polygon++;
    if (disableDrawButton.polygon >= 2) //Reset counter check
    {
        disableDrawButton.polygon = 0;
    }

    if (disableDrawButton.polygon >= 1)//Checks to see if this is our
second click or not
    {
        $('#polygonBtn').addClass('btnSelected'); //Shows the user that
this button is currently selected
        $('#polygonBtn').removeClass('btnNotSelected'); //Shows the
user that this button is currently selected
        setCursorForDrawing();
        swapPolyButtons(1); //Adds polygon confirm button
        var startCoordinates = { x: 0, y: 0 }; //Will change based upon
where the user clicked
        var beginningCoordinates = { x: 0, y: 0 };
        var start = { x: 0, y: 0 };//Used to place polygon start point
on old end point
        var move, outOfBounds = false;
        polyButtons.polyFirstTime = true;
        var polygon;
        drawingData.polygonLineCounter = 0;
        var endCoordinates; //Used in two functions, mouse up and
polygonData completion

        function handleMouseDown()
        {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = size / originalHeight;
            }
            else {
                ratio = size / originalWidth;
            }
            polygon = new createjs.Shape();
            polygon.name = UNCOMPLETED_POLYGON; //Name will become more
specific once the polygon is completed ->This is the name of only one
side, which we will be deleting when we form the whole polygon
            polygon.id = POLYGON;
            polygon.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            stage.addChild(polygon);
            if (polyButtons.polyFirstTime) //If this is a new polygon
            {
                startCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                beginningCoordinates = { x: startCoordinates.x, y:
startCoordinates.y }; //Round the coordinates down for recording so
user can close the polygon easier
            }
            else //Else we are continuing one we've already started
            {
                if (drawingData.redoDrawing.length == 0) {
                    drawingData.undoPoly = false;
                }
                if (drawingData.undoPoly)
                {

                    startCoordinates.x =
(drawingData.polygonData[drawingData.polygonData.length -
1].x[drawingData.polygonData[drawingData.polygonData.length -
1].x.length - 1] * ratio);
                    startCoordinates.y =
(drawingData.polygonData[drawingData.polygonData.length -
1].y[drawingData.polygonData[drawingData.polygonData.length -
1].y.length - 1] * ratio);
                }
                else {
                    startCoordinates.x = start.x;
                    startCoordinates.y = start.y;
                }

            }
            move = true;
        }

        function handleMouseMove() //Shows user where line will be
drawn if they release the mouse ("click up")
        {
            if (move) //Basically - if mouse is down
            {
                polygon.graphics.clear(); //Wipe so old drawing won't
show
                var drawingColor = color.drawingColor();
                var local = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                polygon.graphics.setStrokeStyle(1); //Line width
                polygon.graphics.beginStroke(drawingColor);
                polygon.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
                polygon.graphics.lineTo(local.x, local.y);
                polygon.graphics.endStroke();
                stage.update();
            }
        }

        function handleMouseUp() //Draw final line (user has where they
want it, and thus "lets go"
        {
            if (move) //Basically - if mouse is down
            {
                move = false; //Prevent user from still drawing line
after they release mouse (click "up")
                polygon.graphics.clear(); //Wipe so old drawing won't
show
                endCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY); //Mouse location in local coordinates
                if ((endCoordinates.x == beginningCoordinates.x) &&
(endCoordinates.y == beginningCoordinates.y) &&
!polyButtons.polyFirstTime)
                {
                    completePolygon(); //Auto-complete the polygon for
them if they click where they started, unless first time clicking
                    return; //Get out of this function as
completePolygon did everything for us already
                }
                var drawingColor = color.drawingColor();
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                polygon.graphics.setStrokeStyle(1); //Line width
                polygon.graphics.beginStroke(drawingColor);
                polygon.graphics.moveTo(startCoordinates.x,
startCoordinates.y);
                polygon.graphics.lineTo(endCoordinates.x,
endCoordinates.y);
                polygon.graphics.endStroke();
                /***Grab just the alpha channel as we only need to
record that (drawing color will give us the color)***/
                var drawingColorFill = color.fillColor();
                drawingColorFill =
drawingColorFill.substring(drawingColorFill.indexOf("."));
                drawingColorFill = drawingColorFill.replace(')', '');

/***********************************************************************
**********************************/
                stage.update();
                start.x = endCoordinates.x; //Set the start position of
the next line to the end position of the last line drawn (this line)
                start.y = endCoordinates.y; //Set the start position of
the next line to the end position of the last line drawn (this line)
                if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                {
                    var ratio;
                    if (potraitLayout)
                    {
                        ratio = originalHeight / stage.canvas.height;
                    }
                    else
                    {
                        ratio = originalWidth / stage.canvas.width;
                    }
                    if (polyButtons.polyFirstTime) //Only need to do
the beginning for the first line, then its just the end coordinates
that get pushed to our array
                    {
                        beginningCoordinates.x = beginningCoordinates.x
* ratio; //Update start x for array storage
                        beginningCoordinates.y = beginningCoordinates.y
* ratio; //Update start y for array storage
                    }
                    endCoordinates.x = endCoordinates.x * ratio;
                    endCoordinates.y = endCoordinates.y * ratio;
                }

                if (polyButtons.polyFirstTime) //Create a new polygon
if we're making a new polygon
                {
                    //label is null because the user has yet to set it,
pass the line object so I can remove it later
                    var newPolygon = new polygonConstr(null,
drawingColor, drawingColorFill, polygon, beginningCoordinates.x,
beginningCoordinates.y, endCoordinates.x, endCoordinates.y); //Create
new polygon object
                    drawingData.undoDrawingOrder.push(POLYGON);
//Pushing a POLYGON means the polygonTool was used, so if we undo any
drawings, we undo from the polygonData array
                    drawingData.polygonData.push(newPolygon);
                    drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
                }
                else //We're continuing a polygon, just push end x,y
points ->The last one in the array is the one we're working on
                {

drawingData.polygonData[drawingData.polygonData.length -
1].x.push(endCoordinates.x); //Push the end coordinates of the line to
our polygon object as we already know the beginning coordinates (last
line's endCoordindates)

drawingData.polygonData[drawingData.polygonData.length -
1].y.push(endCoordinates.y); //Push the end coordinates of the line to
our polygon object as we already know the beginning coordinates (last
line's endCoordindates)
                    drawingData.undoDrawingOrder.push(POLYGON);
//Pushing a POLYGON means the polygonTool was used, so if we undo any
drawings, we undo from the polygonData array
                    drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
                }


                polyButtons.polyFirstTime = false; //No longer making
the begining of a polygon
```

```
            }
        }

        function completePolygon()
        {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = stage.canvas.height / originalHeight;
            }
            else {
                ratio = stage.canvas.width / originalWidth;
            }
            polyButtons.polyFirstTime = true; //No longer making the
begining of a polygon
            drawCompletePolygon(true, ratio); // true means we are only
completing the last polygon drawn (this one only and not all of them)
        }

        $("#myCanvas").mousedown(function ()
        {
            if (currentLabelBeingShown())//Checks to see if this is our
second click or not
            {
                handleMouseDown();
            }
            else
            {
                move = false;
            }
        });
        $("#myCanvas").mousemove(function () { handleMouseMove(); });
        $("#myCanvas").mouseup(function () { handleMouseUp(); });
        $("#myCanvas").mouseout(function () { handleMouseUp(); });
        $("#completePolygon").click(function () { completePolygon();
});
    }
}
function drawCircle()
{
    var disableValue = disableDrawButton.circle; //Temp variable that
will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.circle = disableValue; //place temp value back in
    disableDrawButton.circle++;
    if (disableDrawButton.circle >= 2) //Reset counter check
    {
        disableDrawButton.circle = 0;
    }

    if (disableDrawButton.circle >= 1)//Checks to see if this is our
second click or not
    {
        $('#circleBtn').addClass('btnSelected'); //Shows the user that
this button is currently selected
```

```
        $('#circleBtn').removeClass('btnNotSelected'); //Shows the user
that this button is currently selected
        swapPolyButtons(3); //Remove poly buttons
        setCursorForDrawing();
        var startCoordinates = { x: 0, y: 0 };
        var circle;
        var move = false;

        function handleMouseDown()
        {
            move = true;
            startCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY);
            circle = new createjs.Shape();
            circle.name = Date.now();
            circle.id = CIRCLE;
            circle.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            stage.addChild(circle);
        }
        function handleMouseMove()
        {
            if (move)
            {
                var endCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY) ;
                //A^2 + B^2 = C^2
                var radius = Math.sqrt(((endCoordinates.x -
startCoordinates.x) * (endCoordinates.x - startCoordinates.x)) +
((endCoordinates.y - startCoordinates.y) * (endCoordinates.y -
startCoordinates.y)))
                circle.graphics.clear(); //Wipe so old drawing won't
show
                var drawingColor = color.drawingColor();
                var drawingFillAlpha = color.fillColor();
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                circle.graphics.setStrokeStyle(1); //Line width
                circle.graphics.beginStroke(drawingColor);
                circle.graphics.beginFill(drawingFillAlpha);
                circle.graphics.drawCircle(startCoordinates.x,
startCoordinates.y, radius);
                circle.graphics.endStroke();
                stage.update();
             }
        }
        function handleMouseUp() {
            if (move)
            {
                move = false;
                var endCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY);
                //A^2 + B^2 = C^2
                var radius = Math.sqrt(((endCoordinates.x -
startCoordinates.x) * (endCoordinates.x - startCoordinates.x)) +
((endCoordinates.y - startCoordinates.y) * (endCoordinates.y -
startCoordinates.y)))
                circle.graphics.clear(); //Wipe so old drawing won't
show
                if (startCoordinates.x - radius < 0 ||
startCoordinates.x + radius > stage.canvas.width || startCoordinates.y
- radius < 0 || startCoordinates.y + radius > stage.canvas.height)//
Circle is out of bounds
                {
                    alertify.error("Please keep your circle within the
window bounds. Thank you.");
                    stage.update();
                    return;
                }
                var drawingColor = color.drawingColor();
                var drawingFillAlpha = color.fillColor();
                drawingColor = rgb2hex(drawingColor); //Convert rgb to
hex and store color as a hex value
                circle.graphics.setStrokeStyle(1); //Line width
                circle.graphics.beginStroke(drawingColor);
                circle.graphics.beginFill(drawingFillAlpha);
                circle.graphics.drawCircle(startCoordinates.x,
startCoordinates.y, radius);
                circle.graphics.endStroke();
                stage.update();

                /***Grab just the alpha channel as we only need to
record that (drawing color will give us the color)***/
                drawingFillAlpha =
drawingFillAlpha.substring(drawingFillAlpha.indexOf("."));
                drawingFillAlpha = drawingFillAlpha.replace(')', '');

/*********************************************************************
*********************************/

                if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
                {
                    var ratio;
                    if (potraitLayout) {
                        ratio = originalHeight / stage.canvas.height;
                    }
                    else {
                        ratio = originalWidth / stage.canvas.width;
                    }
                    startCoordinates.x = startCoordinates.x * ratio;
//Update start x for array storage
                    startCoordinates.y = startCoordinates.y * ratio;
//Update start y for array storage
                    radius = radius * ratio;
                }

                var newCircle = new circleConstr(null, drawingColor,
drawingFillAlpha, circle, startCoordinates.x, startCoordinates.y,
radius); //label is null because the user has yet to set it, pass the
circle object so I can remove it later (stage.removeChild)
```

```
                drawingData.undoDrawingOrder.push(CIRCLE); //Pushing a
LINE means the lineTool was used, so if we undo any drawings, we undo
from the lineData array
                drawingData.circleData.push(newCircle);
                drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
                stage.update();
            }
        }
        $("#myCanvas").mousedown(function ()
        {
            if (currentLabelBeingShown())//Checks to see if this is our
second click or not
            {
                handleMouseDown();
            }
            else
            {
                move = false;
            }
        });
        $("#myCanvas").mousemove(function () { handleMouseMove(); });
        $("#myCanvas").mouseup(function () { handleMouseUp(); });
        $("#myCanvas").mouseout(function () { handleMouseUp(); });

/***********************************************************************
************************************************************************
**************************************************/
    }

}
function drawAuto_Cluster() {
    var disableValue = disableDrawButton.auto_cluster; //Temp variable
that will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.auto_cluster = disableValue; //place temp value
back in
    disableDrawButton.auto_cluster++;
    if (disableDrawButton.auto_cluster >= 2) //Reset counter check
    {
        disableDrawButton.auto_cluster = 0;
    }

    if (disableDrawButton.auto_cluster >= 1)//Checks to see if this is
our second click or not
    {
        //This is where our code for auto_cluster would go if we had it
implemented
        swapPolyButtons(3); //Remove poly buttons
        alertify.error("This feature will be implemented in the future.
Thank you for your patience.");
        //setCursorForDrawing(); //may or may not need this when
actually implemented
    }
```

```
}
function drawFlood_Fill() {
    var disableValue = disableDrawButton.flood_fill; //Temp variable
that will maintain the value of the object after it gets reset
    turnOffButton(); //Resets all the buttons to LOOK like they aren't
selected AND turns off the button (turns off mouse events that are
triggered by that button)
    disableDrawButton.flood_fill = disableValue; //place temp value
back in
    disableDrawButton.flood_fill++;
    if (disableDrawButton.flood_fill >= 2) //Reset counter check
    {
        disableDrawButton.flood_fill = 0;
    }

    if (disableDrawButton.flood_fill >= 1)//Checks to see if this is
our second click or not
    {
        //This is where our code for auto_cluster would go if we had it
implemented
        swapPolyButtons(3); //Remove poly buttons
        alertify.error("This feature will be implemented in the future.
Thank you for your patience.");
      //  setCursorForDrawing(); //may or may not need this when
actually implemented
    }
}
/**********************************************************/

/****Used in drawing polylines and polygons****/
function swapPolyButtons(id) {
    if (id == 1) //Clicked polygon button
    {
        //Show polygon button
        $("#completePolygon").addClass("show");
        $("#completePolygon").removeClass("hide");
        //Hide polyline button
        $("#polylineConfirmation").addClass("hide");
        $("#polylineConfirmation").removeClass("show");
    }
    else if (id == 2)//Clicked polyline button
    {
        //Show polyline button
        $("#polylineConfirmation").addClass("show");
        $("#polylineConfirmation").removeClass("hide");
        //Hide polygon button
        $("#completePolygon").addClass("hide");
        $("#completePolygon").removeClass("show");
    }
    else //Any button but polygon/polyline
    {
        $("#polylineConfirmation").addClass("hide");
        $("#polylineConfirmation").removeClass("show");
        $("#completePolygon").addClass("hide");
        $("#completePolygon").removeClass("show");
    }
}
```

```
/***********************************************/

/****Disables Tool Selector buttons and all else that should be
disabled when picking a new tool****/
function turnOffButton() {

    /****************Button Style******************/
    //Makes the button look like it is turned off
    if ($('#pointBtn').hasClass('btnSelected'))
    {
        $('#pointBtn').removeClass('btnSelected');
        $('#pointBtn').addClass('btnNotSelected');
    }
    if ($('#pencilBtn').hasClass('btnSelected'))
    {
        $('#pencilBtn').removeClass('btnSelected');
        $('#pencilBtn').addClass('btnNotSelected');
    }
    if ($('#lineBtn').hasClass('btnSelected'))
    {
        $('#lineBtn').removeClass('btnSelected');
        $('#lineBtn').addClass('btnNotSelected');
    }
    if ($('#polylineBtn').hasClass('btnSelected'))
    {
        $('#polylineBtn').removeClass('btnSelected');
        $('#polylineBtn').addClass('btnNotSelected');
    }
    if ($('#polygonBtn').hasClass('btnSelected'))
    {
        $('#polygonBtn').removeClass('btnSelected');
        $('#polygonBtn').addClass('btnNotSelected');
    }
    if ($('#circleBtn').hasClass('btnSelected'))
    {
        $('#circleBtn').removeClass('btnSelected');
        $('#circleBtn').addClass('btnNotSelected');
    }
    if ($('#auto_clusterBtn').hasClass('btnSelected'))
    {
        $('#auto_clusterBtn').removeClass('btnSelected');
        $('#auto_clusterBtn').addClass('btnNotSelected');
    }
    if ($('#flood_fillBtn').hasClass('btnSelected'))
    {
        $('#flood_fillBtn').removeClass('btnSelected');
        $('#flood_fillBtn').addClass('btnNotSelected');
    }
    /***************************************************/
    /*********Mouse Events*********/
    //Actually turns off the button
    $('#myCanvas').off("click");
    $('#myCanvas').off("mousedown");
    $('#myCanvas').off("mouseup");
    $('#myCanvas').off("mousemove");
    $('#myCanvas').off("mouseout");
    $("#completePolygon").off("click");
    $("#polylineConfirmation").off("click");
    /********************************/

    setCursorForDragging();
    swapPolyButtons(3);
    removeTooltip(); //Remove tooltip from selected drawing
    unSelectedObject(); //unselect currently selected object (if one is
selected)
    disableDrawButton.reset(); //Reset button click again counter
(happens every time a new button is clicked)
    polyButtons.reset(); //reset values
    //Don't know which one is showing so hide both - should be faster
than checking
    polyButtons.polygonSelected = false;
    polyButtons.polylineSelected = false;
    if (drawingData.polygonData.length > 0) {
        if (drawingData.polygonData[drawingData.polygonData.length -
1].polygon.name == UNCOMPLETED_POLYGON) //If we have an uncompleted
polygon on the image, complete it
        {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = size / originalHeight;
            }
            else {
                ratio = size / originalWidth;
            }
            drawCompletePolygon(true, ratio);
        }
    }
    if (drawingData.polylineData.length > 0) {
        if (drawingData.polylineData[drawingData.polylineData.length -
1].polyline.name == UNCOMPLETED_POLYLINE) //If we have an uncompleted
polyline on the image, complete it
        {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = size / originalHeight;
            }
            else {
                ratio = size / originalWidth;
            }
            drawCompletePolyline(true, ratio);
        }
    }


}
/**************************************/

/*****Label all the drawings that don't have a label yet****/
function label(e) {
    e.preventDefault();
    if (currentLabelBeingShown())//Checks to see if this is our second
click or not
```

```
    {
        turnOffButton(); //Resets all the buttons to LOOK like they
aren't selected AND turns off the button (turns off mouse events that
are triggered by that button)
        var label;
        if ($("#labelDropDown").hasClass('hide'))
        {
            label = document.getElementById("label");
        }
        else
        {
            var labelDropDownMenu =
document.getElementById("labelDropDown");
            label =
labelDropDownMenu.options[labelDropDownMenu.selectedIndex]; //returns
the option names in the dropdown menu
        }

        if (label.value == "") //If the label field is empty
        {
            alertify.error("You cannot enter nothing for a label.
Please enter a label.");
        }
        else //If the user actually put something in there
        {
            var testLabel;
            testLabel = label.value.replace(/[^a-z'_0-9 ]/ig, "");
//Allow Letters A - z, ', _, numbers 0 - 9, and nothing else
            if (testLabel != label.value) //Bad data
            {
                alertify.alert("Labels can only contain letters,
numbers, apostrophes, and underscores. Please try again.");
                alertify.error("Label failed.");
            }
            else {
                if ($("#labelDropDown").hasClass('hide')) {
                    label = document.getElementById("label");
                }
                else {
                    var labelDropDownMenu =
document.getElementById("labelDropDown");
                    label =
labelDropDownMenu.options[labelDropDownMenu.selectedIndex].value;
//returns the option names in the dropdown menu
                }
                var addedLabel = true;
                //For each drawing that does not have a label, give it
the label the user just entered.
                if (drawingData.pointData.length ||
drawingData.pencilData.length || drawingData.lineData.length ||
drawingData.polylineData.length ||
                    drawingData.polygonData.length ||
drawingData.circleData.length || drawingData.auto_clusterData.length ||
drawingData.flood_fillData.length > 0) //Something has been drawn
                {
                    var labeledSomething = addLabelToDrawings(label);
//Labels everything that hasn't been labeled yet
```

```
                    if (!labeledSomething) {
                        addedLabel = false;
                        alertify.error("All drawings already have a
label attached to them.");
                    }
                }
                else //Nothing has been drawn
                {
                    addedLabel = false;
                    alertify.error("You must draw something before you
can label it.");
                }
                /****Add new label to the dropdown menu if it does not
already exist****/
                if (addedLabel) {
                    addLabelToDropDownMenu(label);
                    alertify.success("Label added successfully to all
shapes that previously didn't have a label.");
                }

/***********************************************************************
*/
            }

        }
    }
    //Clear the label field
    if ($("#labelDropDown").hasClass('hide')) {
        label.value = "";
    }
}
/************************************************************/

function addLabelToDropDownMenu(label) //Add new label to the dropdown
menu if it does not already exist
{
    var dropDownMenu = document.getElementById("dropDown");
    var labelFound = false;
    for (var i = 0; i < dropDownMenu.length; i++) //Search for the
label in the drop down menu
    {
        var optionValues = dropDownMenu.options[i].text; //returns the
option names in the dropdown menu
        if (optionValues == label.value || optionValues == label) {
            labelFound = true;
        }
    }
    if (!labelFound) //If the label wasn't found, add it to the
dropdown menu
    {
        var option = document.createElement("option");
        option.text = label.value || label;
        dropDownMenu.add(option);
    }
}

function addLabelToDrawings(label)
```

```
{
    label = label.value || label; //Depending on where the function
gets called from, this will determine which label is correct and which
one is undefined
    var labeledSomething = false;
    var labelCount = 0;
    //Point Tool
    for (var i = 0; i < drawingData.pointData.length; i++) //If the
array isn't empty
    {
        if (drawingData.pointData[i].label == null) //If it hasn't been
assigned a label yet
        {
            drawingData.pointData[i].label = label; //Assign the point
the label that the user entered
            labelCount++;
            labeledSomething = true;
        }
    }
    //Pencil Tool
    for (var i = 0; i < drawingData.pencilData.length; i++) //If the
array isn't empty
    {
        if (drawingData.pencilData[i].label == null) //If it hasn't
been assigned a label yet
        {
            drawingData.pencilData[i].label = label; //Assign the point
the label that the user entered
            labelCount++;
            labeledSomething = true;
        }
    }
    //Line Tool
    for (var i = 0; i < drawingData.lineData.length; i++) //If the
array isn't empty
    {
        if (drawingData.lineData[i].label == null) //If it hasn't been
assigned a label yet
        {
            drawingData.lineData[i].label = label; //Assign the point
the label that the user entered
            labelCount++;
            labeledSomething = true;
        }
    }
    //Polyline Tool
    for (var i = 0; i < drawingData.polylineData.length; i++) //If the
array isn't empty
    {
        if (drawingData.polylineData[drawingData.polylineData.length -
1].polyline.name == UNCOMPLETED_POLYLINE) //If we have an uncompleted
polyline
        {
            polyButtons.polyFirstTime = true; //No longer making the
begining of a polyline
            drawCompletePolyline(true); //Auto-complete the polyline
        }
        if (drawingData.polylineData[i].label == null) //If it hasn't
been assigned a label yet
        {
            drawingData.polylineData[i].label = label; //Assign the
point the label that the user entered drawingData.
            labelCount++;
            labeledSomething = true;
        }
    }
    //Polygon Tool
    for (var i = 0; i < drawingData.polygonData.length; i++) //If the
array isn't empty
    {
        if (drawingData.polygonData[drawingData.polygonData.length -
1].polygon.name == UNCOMPLETED_POLYGON) //If we have an uncompleted
polygon
        {
            polyButtons.polyFirstTime = true; //No longer making the
begining of a polygon
            drawCompletePolygon(true); //Auto-complete the polygon
        }
        if (drawingData.polygonData[i].label == null) //If it hasn't
been assigned a label yet
        {
            drawingData.polygonData[i].label = label; //Assign the
point the label that the user entered drawingData.
            labelCount++;
            labeledSomething = true;
        }
    }
    //Circle Tool
    for (var i = 0; i < drawingData.circleData.length; i++) //If the
array isn't empty
    {
        if (drawingData.circleData[i].label == null) //If it hasn't
been assigned a label yet
        {
            drawingData.circleData[i].label = label; //Assign the point
the label that the user entered
            labelCount++;
            labeledSomething = true;
        }
    }
    //Auto-Cluster Tool
    for (var i = 0; i < drawingData.auto_clusterData.length; i++) //If
the array isn't empty
    {
        if (drawingData.auto_clusterData[i].label == null) //If it
hasn't been assigned a label yet
        {
            drawingData.auto_clusterData[i].label = label; //Assign the
point the label that the user entered
            labelCount++;
            labeledSomething = true;
        }
    }
    //Flood-Fill Tool
    for (var i = 0; i < drawingData.flood_fillData.length; i++) //If
the array isn't empty
    {
        if (drawingData.flood_fillData[i].label == null) //If it hasn't
been assigned a label yet
        {
            drawingData.flood_fillData[i].label = label; //Assign the
point the label that the user entered
            labelCount++;
            labeledSomething = true;
        }
    }
    //Label adder
    var dropDownMenu = document.getElementById("dropDown");
    var found = false;
    var positionToAddMore;
    for (var i = 1; i < dropDownMenu.length; i++) //Cycle through the
dropdown menu, looking at option values start at one to skip "All
Labels"
    {
        for (var j = 0; j < drawingData.labelCount.length; j++) //Go
through label array seeing if we already have this one
        {
            if (label == drawingData.labelCount[j]) //If we have it,
say so
            {
                positionToAddMore = j;
                found = true;
                break;
            }
            found = false;
        }
    }
    if (!found) //If we didn't find a match, push new label nd its
count to arary
    {
        drawingData.labelCount.push(label, labelCount);
    }
    else {
        drawingData.labelCount[positionToAddMore + 1] += labelCount;
//Update how many shapes have that label
    }

    return labeledSomething;

}

function resizeRedraw(ratio)
{
    removeTooltip();//Remove tooltip
    unSelectedObject(); //unselect currently selected object (if one is
selected)
    if (drawingData.pointData.length > 0) //If there were any points
drawn on the map
    {
        for (var i = 0; i < drawingData.pointData.length; i++) //For
every previously drawn point, draw again with the same style
```

```
        {
            var point = new createjs.Shape();
            point.name = Math.floor(Date.now() * Math.random());
            point.id = POINT;
            point.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            point.graphics.setStrokeStyle(.25);
            point.graphics.beginFill(drawingData.pointData[i].color);
            point.graphics.beginStroke(drawingData.pointData[i].color);
            point.graphics.drawRect((drawingData.pointData[i].x *
ratio), (drawingData.pointData[i].y * ratio), 1, 1); //Scales x,y to
new location and places 1 pixel dot there
            drawingData.pointData[i].point = point;
            stage.addChild(drawingData.pointData[i].point);
        }
    }
    if (drawingData.pencilData.length > 0) //If there were any lines
drawn on the map
    {
        for (var i = 0; i < drawingData.pencilData.length; i++) //For
every previously drawn line, draw again with the same style
        {
            var pencil = new createjs.Shape();
            pencil.name = Math.floor(Date.now() * Math.random());
            pencil.id = PENCIL;
            pencil.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            pencil.graphics.setStrokeStyle(1);

pencil.graphics.beginStroke(drawingData.pencilData[i].color)
            pencil.graphics.moveTo(drawingData.pencilData[i].x[0] *
ratio, drawingData.pencilData[i].y[0] * ratio); //0 is our first point,
thus start at 1 for j (don't repeat)
            for (var j = 1; j < drawingData.pencilData[i].x.length;
j++)
            {
                pencil.graphics.lineTo(drawingData.pencilData[i].x[j] *
ratio, drawingData.pencilData[i].y[j] * ratio);
            }
            pencil.graphics.endStroke();
            drawingData.pencilData[i].pencil = pencil;
            stage.addChild(drawingData.pencilData[i].pencil);
        }
    }
    if (drawingData.lineData.length > 0) //If there were any lines
drawn on the map
    {
        for (var i = 0; i < drawingData.lineData.length; i++) //For
every previously drawn line, draw again with the same style
        {
            var line = new createjs.Shape();
            line.name = Math.floor(Date.now() * Math.random());
            line.id = LINE;
            line.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            line.graphics.setStrokeStyle(1);
            line.graphics.beginStroke(drawingData.lineData[i].color)
            //Use 0 and 1 as array locations because a line will only
have two in its array and this is the fastest way of grabbing those two
            line.graphics.moveTo(drawingData.lineData[i].x[0] * ratio,
drawingData.lineData[i].y[0] * ratio);
            line.graphics.lineTo(drawingData.lineData[i].x[1] * ratio,
drawingData.lineData[i].y[1] * ratio);
            line.graphics.endStroke();
            drawingData.lineData[i].lineObject = line;
            stage.addChild(drawingData.lineData[i].lineObject);
        }
    }
    if (drawingData.polylineData.length > 0) //If there were any lines
drawn on the map
    {
        drawCompletePolyline(false, ratio); //redraws all polylines
    }
    if (drawingData.polygonData.length > 0) //If there were any lines
drawn on the map
    {
        drawCompletePolygon(false, ratio); //redraws all polygons
    }
    if (drawingData.circleData.length > 0)
    {
        for (var i = 0; i < drawingData.circleData.length; i++)
        {
            var circle = new createjs.Shape();
            circle.name = Math.floor(Date.now() * Math.random());
            circle.id = CIRCLE;
            circle.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            circle.graphics.setStrokeStyle(1);

circle.graphics.beginStroke(drawingData.circleData[i].color)

circle.graphics.beginFill(hex2rgba(drawingData.circleData[i].color,
drawingData.circleData[i].fillColor));
            circle.graphics.drawCircle(drawingData.circleData[i].x *
ratio, drawingData.circleData[i].y * ratio,
drawingData.circleData[i].radius * ratio);
            circle.graphics.endStroke();
            drawingData.circleData[i].circle = circle;
            stage.addChild(drawingData.circleData[i].circle);
        }
    }
}


function gatherServerData(serverData) //This is how the file we
download and send to C++ is created
```

```
{
    for (var i = 0; i < drawingData.pointData.length; i++) //Loops
through our point data array and adds to our data string
    {
        serverData += drawingData.pointData[i].id + ','
            + drawingData.pointData[i].label + ','
            + drawingData.pointData[i].color + ','
            + drawingData.pointData[i].x + ','
            + drawingData.pointData[i].y + ',,\\n'; //Extra comma
padding for Coordinate Selector application.
        drawingCount++;
    }
    for (var i = 0; i < drawingData.pencilData.length; i++) //Loops
through our pencilData array and adds to our data string
    {
        serverData += drawingData.pencilData[i].id + ','
            + drawingData.pencilData[i].label + ','
            + drawingData.pencilData[i].color;
        for (var j = 0; j < drawingData.pencilData[i].x.length; j++)
//Start and end coordinates of the line
        {
            serverData += ',' + drawingData.pencilData[i].x[j] + ','
                        + drawingData.pencilData[i].y[j];
        }
        if (i == drawingData.pencilData.length - 1)
        {
            serverData += ',,\\n'; //Extra comma padding for Coordinate
Selector application.
        }
        else
        {
            serverData += '\\n';
        }
        drawingCount++;
    }
    for (var i = 0; i < drawingData.lineData.length; i++) //Loops
through our lineData array and adds to our data string
    {
        serverData += drawingData.lineData[i].id + ','
            + drawingData.lineData[i].label + ','
            + drawingData.lineData[i].color;
        for (var j = 0; j < drawingData.lineData[i].x.length; j++)
//Start and end coordinates of the line
        {
            serverData += ',' + drawingData.lineData[i].x[j] + ','
                       + drawingData.lineData[i].y[j];
        }
        if (i == drawingData.lineData.length - 1)
        {
            serverData += ',,\\n'; //Extra comma padding for Coordinate
Selector application.
        }
        else
        {
            serverData += '\\n';
        }
        drawingCount++;
    }
    for (var i = 0; i < drawingData.polylineData.length; i++) //Loops
through our polygonData array and adds to our data string
    {
        serverData += drawingData.polylineData[i].id + ','
            + drawingData.polylineData[i].label + ','
            + drawingData.polylineData[i].color;
        for (var j = 0; j < drawingData.polylineData[i].x.length; j++)
{
            serverData += ',' + drawingData.polylineData[i].x[j] + ','
           + drawingData.polylineData[i].y[j];

        }
        if (i == drawingData.polyline.length - 1)
        {
            serverData += ',,\\n'; //Extra comma padding for Coordinate
Selector application.
        }
        else
        {
            serverData += '\\n';
        }
        drawingCount++;
    }
    for (var i = 0; i < drawingData.polygonData.length; i++) //Loops
through our polygonData array and adds to our data string
    {
        serverData += drawingData.polygonData[i].id + ','
            + drawingData.polygonData[i].label + ','
            + drawingData.polygonData[i].color + ','
            + drawingData.polygonData[i].fillColor;
        for (var j = 0; j < drawingData.polygonData[i].x.length; j++)
        {
            serverData += ',' + drawingData.polygonData[i].x[j] + ','
           + drawingData.polygonData[i].y[j];

        }
        if (i == drawingData.polygonData.length - 1)
        {
            serverData += ',,\\n'; //Extra comma padding for Coordinate
Selector application.
        }
        else
        {
            serverData += '\\n';
        }
        drawingCount++;

    }
    for (var i = 0; i < drawingData.circleData.length; i++) //Loops
through our circleData array and adds to our data string
    {
        serverData += drawingData.circleData[i].id + ','
            + drawingData.circleData[i].label + ','
            + drawingData.circleData[i].color + ','
            + drawingData.circleData[i].fillColor + ','
            + drawingData.circleData[i].x + ','
            + drawingData.circleData[i].y + ','
            + drawingData.circleData[i].radius + ',,\\n';
        drawingCount++;
    }
    for (var i = 0; i < drawingData.auto_clusterData.length; i++)
//Loops through our auto_clusterData array and adds to our data string
    {
        //serverData += auto_clusterData stuff
        drawingCount++;
    }
    for (var i = 0; i < drawingData.flood_fillData.length; i++) //Loops
through our flood_fillData array and adds to our data string
    {
        //ServerData += flood_fillData stuff
        drawingCount++;
    }
    return serverData;
}

function labelCheck () //Check to see if all drawings have a label
before saving any of them
{
    if (drawingData.pointData.length > 0) //Check pointData Array
    {
        if (drawingData.pointData[drawingData.pointData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.pencilData.length > 0) //Check pencilData Array
    {
        if (drawingData.pencilData[drawingData.pencilData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.lineData.length > 0) //Check lineData array
    {
        if (drawingData.lineData[drawingData.lineData.length - 1].label
== null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.polylineData.length > 0) //check polyline array
    {
        if (drawingData.polylineData[drawingData.polylineData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.polygonData.length > 0) //check polygon array
    {
        if (drawingData.polygonData[drawingData.polygonData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.circleData.length > 0) //check circle array
    {
        if (drawingData.circleData[drawingData.circleData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.auto_clusterData.length > 0) //check auto_cluster
array
    {
        if
(drawingData.auto_clusterData[drawingData.auto_clusterData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    if (drawingData.flood_fillData.length > 0) //check flood_fill array
    {
        if
(drawingData.flood_fillData[drawingData.flood_fillData.length -
1].label == null)
        {
            return false; //Last point isn't labeled -> not all
drawings are labeled -> Can't save
        }
    }
    return true;
}

function undo()
{
    if (currentLabelBeingShown())//Checks to see if this is our second
click or not
    {
        if (drawingData.undoDrawingOrder.length > 0) //If we have any
drawings
        {
            removeTooltip();//Remove tooltip
            unSelectedObject(); //unselect currently selected object
(if one is selected)
            if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== POINT) //If the last thing that was drawn was a point
```

```
            {
                lastLabelCheck(true, POINT); //True because that will
trigger the undo side of the function (subtract a label)

stage.removeChild(drawingData.pointData[drawingData.pointData.length -
1].point);
                stage.update();

drawingData.redoDrawing.push(drawingData.pointData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== PENCIL) //If the last thing that was drawn was a pencil
            {
                lastLabelCheck(true, PENCIL); //True because that will
trigger the undo side of the function (subtract a label)

stage.removeChild(drawingData.pencilData[drawingData.pencilData.length
- 1].pencil);
                stage.update();

drawingData.redoDrawing.push(drawingData.pencilData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== LINE) //If the last thing that was drawn was a line
            {
                lastLabelCheck(true, LINE); //True because that will
trigger the undo side of the function (subtract a label)

stage.removeChild(drawingData.lineData[drawingData.lineData.length -
1].lineObject);
                stage.update();

drawingData.redoDrawing.push(drawingData.lineData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());

            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== POLYLINE) //If the last thing that was drawn was a polyline
            {
                var ratio;
                var size = document.getElementById("size").value;
                if (potraitLayout) {
                    ratio = size / originalHeight;
                }
                else {
                    ratio = size / originalWidth;
                }
                if
(drawingData.polylineData[drawingData.polylineData.length -
1].polyline.name == UNCOMPLETED_POLYLINE) //if an uncompletedpolyline -
>undo just a line
                {
                    drawingData.undoPoly = true; //Starts the mouse
from the last drawn vertex of the polyline (the point before this one)-
> not needed if polyline is already completed (thus not in other
statement)
                    //Remove all lines, then redraw them except for the
last one
                    var test =
drawingData.polylineData[drawingData.polylineData.length - 1].x.length;
                    if (test > 2) //if we aren't about to remove the
last line
                    {
                        drawingData.undoPoly = true;
                        var polyline = new createjs.Shape();
                        polyline.name = UNCOMPLETED_POLYLINE;
                        polyline.id = POLYLINE;
                        polyline.shadow = new createjs.Shadow("#000",
0, 0, 0); //Create a new shadow that will display if the user selects
this shape after creating it to show the user that this shape is
currently selected
                        var numberOfChildren = stage.getNumChildren();
                        for (var i = numberOfChildren - 1; i > 0; i--)
{
                            var child = stage.getChildAt(i);
                            if (UNCOMPLETED_POLYLINE == child.name)
//if an uncompleted polyline line segment is found, remove it and say
we have a new polyline to be drawn
                            {
                                stage.removeChildAt(i);
                                stage.update();
                            }
                        }

drawingData.redoDrawing.push(drawingData.polylineData[drawingData.polyl
ineData.length - 1].x.pop()); //push old x to redoDrawing stack

drawingData.redoDrawing.push(drawingData.polylineData[drawingData.polyl
ineData.length - 1].y.pop()); //push old  y to redoDrawing stack

polyline.graphics.beginStroke(drawingData.polylineData[drawingData.poly
lineData.length - 1].color);

polyline.graphics.moveTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[0] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
                        //Loop through all the points for this polyline
and draw them
                        var test =
drawingData.polylineData[drawingData.polylineData.length - 1].x.length;
                        for (var i = 1; i < test; i++) {

polyline.graphics.lineTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[i] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[i] *
ratio)); //Draw to the next point on the polygon

                        }
                        polyline.graphics.endStroke();

drawingData.polylineData[drawingData.polylineData.length - 1].polyline
= polyline;

stage.addChild(drawingData.polylineData[drawingData.polylineData.length
- 1].polyline);
                        stage.update();

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                    }
                    else //if we're about to remove the last line
segment of the polyline
                    {

stage.removeChild(drawingData.polylineData[drawingData.polylineData.len
gth - 1].polyline);
                        stage.update();

drawingData.redoDrawing.push(drawingData.polylineData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                        polyButtons.polyFirstTime = true;
                    }
                }
                else //undo completed polyline
                {
                    lastLabelCheck(true, POLYLINE); //True because that
will trigger the undo side of the function (subtract a label) //Only
completed polygons will have a label, thus only needed here

stage.removeChild(drawingData.polylineData[drawingData.polylineData.len
gth - 1].polyline);
                    stage.update();

drawingData.redoDrawing.push(drawingData.polylineData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                }
            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== POLYGON) //If the last thing that was drawn was a  polygon
            {
                var ratio;
                var size = document.getElementById("size").value;
                if (potraitLayout) {
                    ratio = size / originalHeight;
                }
                else {
                    ratio = size / originalWidth;
                }
                if
(drawingData.polygonData[drawingData.polygonData.length -
1].polygon.name == UNCOMPLETED_POLYGON) //if an uncompletedpolygon -
>undo just a line
                {
                    drawingData.undoPoly = true; //Starts the mouse
from the last drawn vertex of the polygon (the point before this one)->
not needed if polygon is already completed (thus not in other
statement)
                    //Remove all lines, then redraw them except for the
last one
                    var test =
drawingData.polygonData[drawingData.polygonData.length - 1].x.length;
                    if (test > 2) //if we aren't about to remove the
last line
                    {
                        drawingData.undoPoly = true;
                        var polygon = new createjs.Shape();
                        polygon.name = UNCOMPLETED_POLYGON;
                        polygon.id = POLYGON;
                        polygon.shadow = new createjs.Shadow("#000", 0,
0, 0); //Create a new shadow that will display if the user selects this
shape after creating it to show the user that this shape is currently
selected
                        var numberOfChildren = stage.getNumChildren();
                        for (var i = numberOfChildren - 1; i > 0; i--)
{
                            var child = stage.getChildAt(i);
                            if (UNCOMPLETED_POLYGON == child.name) //if
an uncompleted polygon line segment is found, remove it and say we have
a new polygon to be drawn
                            {
                                stage.removeChildAt(i);
                                stage.update();
                            }
                        }

drawingData.redoDrawing.push(drawingData.polygonData[drawingData.polygo
nData.length - 1].x.pop()); //push old x to redoDrawing stack

drawingData.redoDrawing.push(drawingData.polygonData[drawingData.polygo
nData.length - 1].y.pop()); //push old  y to redoDrawing stack

polygon.graphics.beginStroke(drawingData.polygonData[drawingData.polygo
nData.length - 1].color);

polygon.graphics.moveTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
                        //Loop through all the points for this polygon
and draw them
                        var test =
drawingData.polygonData[drawingData.polygonData.length - 1].x.length;
                        for (var i = 1; i < test; i++) {

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[i] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[i] *
ratio)); //Draw to the next point on the polygon

                        }
                        polygon.graphics.endStroke();

drawingData.polygonData[drawingData.polygonData.length - 1].polygon =
polygon;

stage.addChild(drawingData.polygonData[drawingData.polygonData.length -
1].polygon);
                        stage.update();

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                    }
                    else //if we're about to remove the last line
segment of the polygon
                    {

stage.removeChild(drawingData.polygonData[drawingData.polygonData.lengt
h - 1].polygon);
                        stage.update();

drawingData.redoDrawing.push(drawingData.polygonData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                        polyButtons.polyFirstTime = true;
                    }
                }
                else //undo completed polygon
                {
                    lastLabelCheck(true, POLYGON); //True because that
will trigger the undo side of the function (subtract a label) //Only
completed polygons will have a label, thus only needed here

stage.removeChild(drawingData.polygonData[drawingData.polygonData.lengt
h - 1].polygon);
                    stage.update();

drawingData.redoDrawing.push(drawingData.polygonData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                }
            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== CIRCLE) //If the last thing that was drawn was a circle
            {
                lastLabelCheck(true, CIRCLE); //True because that will
trigger the undo side of the function (subtract a label)

stage.removeChild(drawingData.circleData[drawingData.circleData.length
- 1].circle);
                stage.update();

drawingData.redoDrawing.push(drawingData.circleData.pop());
drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== AUTO_CLUSTER) //If the last thing that was drawn was a auto_cluster
            {
                lastLabelCheck(true, AUTO_CLUSTER); //True because that
will trigger the undo side of the function (subtract a label)

drawingData.redoDrawing.push(drawingData.auto_clusterData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                resize();
            }
            else if
(drawingData.undoDrawingOrder[drawingData.undoDrawingOrder.length - 1]
== FLOOD_FILL) //If the last thing that was drawn was a flood_fill
            {
                lastLabelCheck(true, FLOOD_FILL); //True because that
will trigger the undo side of the function (subtract a label)

drawingData.redoDrawing.push(drawingData.flood_fillData.pop());

drawingData.redoDrawing.push(drawingData.undoDrawingOrder.pop());
                resize();
            }
            else //This else should never trigger.... But you never
know
            {
                alertify.error("We could not find anything to undo.");
            }
        }
        else //Nothing to undo
        {
            alertify.error("You cannot undo further.");
        }
    }
}

function redo() {
    if (currentLabelBeingShown())//Checks to see if this is our second
click or not
    {
        if (drawingData.redoDrawing.length > 0) //If we have any
drawings
        {
            var ratio;
            var size = document.getElementById("size").value;
            if (potraitLayout) {
                ratio = size / originalHeight;
            }
            else {
                ratio = size / originalWidth;
            }
            var drawingID = drawingData.redoDrawing.pop();
            if (drawingID == POINT) //Point
            {

drawingData.pointData.push(drawingData.redoDrawing.pop());
                drawingData.undoDrawingOrder.push(POINT); //Pushing a
POINT means the pointTool was used, so if we undo any drawings, we undo
from the pointData array
                var point = new createjs.Shape();
                point.name = Math.floor(Date.now() * Math.random());
                point.id = POINT;
                point.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
                point.graphics.setStrokeStyle(.25);

point.graphics.beginFill(drawingData.pointData[drawingData.pointData.le
ngth - 1].color);

point.graphics.beginStroke(drawingData.pointData[drawingData.pointData.
length - 1].color);

point.graphics.drawRect((drawingData.pointData[drawingData.pointData.le
ngth - 1].x * ratio),
(drawingData.pointData[drawingData.pointData.length - 1].y * ratio), 1,
1); //Scales x,y to new location and places 1 pixel dot there

stage.addChild(drawingData.pointData[drawingData.pointData.length -
1].point);
                stage.update();
                lastLabelCheck(false, POINT); //False because that will
trigger the redo side of the function (add a label)
            }
            else if (drawingID == PENCIL) //Pencil
            {

drawingData.pencilData.push(drawingData.redoDrawing.pop());
                drawingData.undoDrawingOrder.push(PENCIL); //Pushing a
PENCIL means the pencil tool was used, so if we undo any drawings, we
undo from the pencil array
                var pencil = new createjs.Shape();
                pencil.name = Math.floor(Date.now() * Math.random());
                pencil.id = PENCIL;
                pencil.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
                pencil.graphics.setStrokeStyle(1);

pencil.graphics.beginStroke(drawingData.pencilData[drawingData.pencilDa
ta.length - 1].color)

pencil.graphics.moveTo(drawingData.pencilData[drawingData.pencilData.le
ngth - 1].x[0] * ratio,
drawingData.pencilData[drawingData.pencilData.length - 1].y[0] *
ratio); //0 is our first point, thus start at 1 for j (don't repeat)
```

```
                for (var j = 1; j <
drawingData.pencilData[drawingData.pencilData.length - 1].x.length;
j++) {

pencil.graphics.lineTo(drawingData.pencilData[drawingData.pencilData.le
ngth - 1].x[j] * ratio,
drawingData.pencilData[drawingData.pencilData.length - 1].y[j] *
ratio);
                }
                pencil.graphics.endStroke();
                drawingData.pencilData[drawingData.pencilData.length -
1].pencil = pencil;

stage.addChild(drawingData.pencilData[drawingData.pencilData.length -
1].pencil);
                stage.update();
                lastLabelCheck(false, PENCIL); //False because that
will trigger the redo side of the function (add a label)
            }
            else if (drawingID == LINE) //Line
            {

drawingData.lineData.push(drawingData.redoDrawing.pop());
                drawingData.undoDrawingOrder.push(LINE); //Pushing a
LINE means the line tool was used, so if we undo any drawings, we undo
from the line array
                var line = new createjs.Shape();
                line.name = Math.floor(Date.now() * Math.random());
                line.id = LINE;
                line.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
                line.graphics.setStrokeStyle(1);

line.graphics.beginStroke(drawingData.lineData[drawingData.lineData.len
gth - 1].color)
                //Can do 0 and 1 for xy because there are always only
two points in a line and thus only two data points in our array

line.graphics.moveTo(drawingData.lineData[drawingData.lineData.length -
1].x[0] * ratio, drawingData.lineData[drawingData.lineData.length -
1].y[0] * ratio);

line.graphics.lineTo(drawingData.lineData[drawingData.lineData.length -
1].x[1] * ratio, drawingData.lineData[drawingData.lineData.length -
1].y[1] * ratio);
                line.graphics.endStroke();
                drawingData.lineData[drawingData.lineData.length -
1].lineObject = line;

stage.addChild(drawingData.lineData[drawingData.lineData.length -
1].lineObject);
                stage.update();
                lastLabelCheck(false, LINE); //False because that will
trigger the redo side of the function (add a label)
            }
            else if (drawingID == POLYLINE) {

drawingData.polylineData.push(drawingData.redoDrawing.pop()); //could
be a full polyline or an x or y coordinate
                drawingData.undoDrawingOrder.push(POLYLINE); //Pushing
a POLYLINE means the polyline tool was used, so if we undo any
drawings, we undo from the polyline array
                //if last item in polylineData is a number (x or y) OR
it is an uncompleted polyline (last line segment just was undone from
the polyline)
                if
(drawingData.polylineData[drawingData.polylineData.length - 1].polyline
== null || drawingData.polylineData[drawingData.polylineData.length -
1].polyline.name == UNCOMPLETED_POLYLINE) {
                    if
(drawingData.polylineData[drawingData.polylineData.length - 1].polyline
== null) // if we removed the x,y and not the polyline itself, add x,y
back then redraw, else just redraw polyline (which will be a 1 line
segment)
                    {
                        var yCoordinate =
drawingData.polylineData.pop();

drawingData.polylineData[drawingData.polylineData.length -
1].y.push(yCoordinate); //remove the y coordinate that got pushed to
polylineData

drawingData.polylineData[drawingData.polylineData.length -
1].x.push(drawingData.redoDrawing.pop()); //remove the x coordinate
that got pushed to the redoDrawing array in the undo function
                    }
                    else //We are drawing the first polyline segment,
this is no longer our first time drawing anything
                    {
                        polyButtons.polyFirstTime = false;
                    }
                    var polyline = new createjs.Shape();
                    polyline.name = UNCOMPLETED_POLYLINE;
                    polyline.id = POLYLINE;
                    polyline.shadow = new createjs.Shadow("#000", 0, 0,
0); //Create a new shadow that will display if the user selects this
shape after creating it to show the user that this shape is currently
selected

polyline.graphics.beginStroke(drawingData.polylineData[drawingData.poly
lineData.length - 1].color);
                    //Move to the starting position of the second to
last line

polyline.graphics.moveTo((drawingData.polylineData[drawingData.polyline
Data.length -
1].x[drawingData.polylineData[drawingData.polylineData.length -
1].x.length - 2] * ratio),
(drawingData.polylineData[drawingData.polylineData.length -
1].y[drawingData.polylineData[drawingData.polylineData.length -
1].y.length - 2] * ratio));
```

```
                    //Draw to the next point on the polyline (the
"last" point)
                    
polyline.graphics.lineTo((drawingData.polylineData[drawingData.polyline
Data.length -
1].x[drawingData.polylineData[drawingData.polylineData.length -
1].x.length - 1] * ratio),
(drawingData.polylineData[drawingData.polylineData.length -
1].y[drawingData.polylineData[drawingData.polylineData.length -
1].y.length - 1] * ratio));
                    polyline.graphics.endStroke();
                    
drawingData.polylineData[drawingData.polylineData.length - 1].polyline
= polyline;
                    
stage.addChild(drawingData.polylineData[drawingData.polylineData.length
- 1].polyline);
                    stage.update();
                }
                else {
                    var polyline = new createjs.Shape();
                    
polyline.graphics.beginStroke(drawingData.polylineData[drawingData.poly
lineData.length - 1].color);
                    
polyline.graphics.moveTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[0] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
                    //Loop through all the points for this polyline and
draw them
                    for (var i = 1; i <
drawingData.polylineData[drawingData.polylineData.length - 1].x.length;
i++) {
                        
polyline.graphics.lineTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[i] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[i] *
ratio)); //Draw to the next point on the polygon
                    }
                    polyline.graphics.endStroke();
                    polyline.name = "completedPolyline" + Date.now();
                    polyline.id = POLYLINE;
                    polyline.shadow = new createjs.Shadow("#000", 0, 0,
0); //Create a new shadow that will display if the user selects this
shape after creating it to show the user that this shape is currently
selected
                    
drawingData.polylineData[drawingData.polylineData.length - 1].polyline
= polyline;
                    
stage.addChild(drawingData.polylineData[drawingData.polylineData.length
- 1].polyline);
                    stage.update();
                    lastLabelCheck(false, POLYLINE);
                }
            }
            else if (drawingID == POLYGON) {

drawingData.polygonData.push(drawingData.redoDrawing.pop()); //could be
a full polygon or an x or y coordinate
                drawingData.undoDrawingOrder.push(POLYGON);  //Pushing
a POLYGON means the polygon tool was used, so if we undo any drawings,
we undo from the polygon array
                //if last item in polygonDaya is a number (x or y) OR
it is an uncompleted polygon (last line segment just was undone from
the polygon)
                if
(drawingData.polygonData[drawingData.polygonData.length - 1].polygon ==
null || drawingData.polygonData[drawingData.polygonData.length -
1].polygon.name == UNCOMPLETED_POLYGON) {
                    if
(drawingData.polygonData[drawingData.polygonData.length - 1].polygon ==
null) // if we removed the x,y and not the polygon itself, add x,y back
then redraw, else just redraw polygon (which will be a 1 line segment)
                    {
                        var yCoordinate =
drawingData.polygonData.pop();

drawingData.polygonData[drawingData.polygonData.length -
1].y.push(yCoordinate); //remove the y coordinate that got pushed to
polygonData

drawingData.polygonData[drawingData.polygonData.length -
1].x.push(drawingData.redoDrawing.pop()); //remove the x coordinate
that got pushed to the redoDrawing array in the undo function
                    }
                    else //We are drawing the first polygon segment,
this is no longer our first time drawing anything
                    {
                        polyButtons.polyFirstTime = false;
                    }
                    var polygon = new createjs.Shape();
                    polygon.name = UNCOMPLETED_POLYGON;
                    polygon.id = POLYGON;
                    polygon.shadow = new createjs.Shadow("#000", 0, 0,
0); //Create a new shadow that will display if the user selects this
shape after creating it to show the user that this shape is currently
selected

polygon.graphics.beginStroke(drawingData.polygonData[drawingData.polygo
nData.length - 1].color);
                    //Move to the starting position of the second to
last line

polygon.graphics.moveTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[drawingData.polygonData[drawingData.polygonData.length
- 1].x.length - 2] * ratio),
(drawingData.polygonData[drawingData.polygonData.length -
1].y[drawingData.polygonData[drawingData.polygonData.length -
1].y.length - 2] * ratio));
                    //Draw to the next point on the polygon (the "last"
point)
```

```
polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[drawingData.polygonData[drawingData.polygonData.length
- 1].x.length - 1] * ratio),
(drawingData.polygonData[drawingData.polygonData.length -
1].y[drawingData.polygonData[drawingData.polygonData.length -
1].y.length - 1] * ratio));
                    polygon.graphics.endStroke();

drawingData.polygonData[drawingData.polygonData.length - 1].polygon =
polygon;

stage.addChild(drawingData.polygonData[drawingData.polygonData.length -
1].polygon);
                    stage.update();
                }
                else {
                    var polygon = new createjs.Shape();

polygon.graphics.beginStroke(drawingData.polygonData[drawingData.polygo
nData.length - 1].color);

polygon.graphics.beginFill(hex2rgba(drawingData.polygonData[drawingData
.polygonData.length - 1].color,
drawingData.polygonData[drawingData.polygonData.length -
1].fillColor));

polygon.graphics.moveTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
                    //Loop through all the points for this polygon and
draw them
                    for (var i = 1; i <
drawingData.polygonData[drawingData.polygonData.length - 1].x.length;
i++) {

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[i] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[i] *
ratio)); //Draw to the next point on the polygon
                    }

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Connect last point with the beginning point
                    polygon.graphics.endStroke();
                    polygon.name = "completedPolygon" + Date.now();
                    polygon.id = POLYGON;
                    polygon.shadow = new createjs.Shadow("#000", 0, 0,
0); //Create a new shadow that will display if the user selects this
shape after creating it to show the user that this shape is currently
selected

drawingData.polygonData[drawingData.polygonData.length - 1].polygon =
polygon;
```

```
stage.addChild(drawingData.polygonData[drawingData.polygonData.length -
1].polygon);
                    stage.update();
                    lastLabelCheck(false, POLYGON);
                }
            }
            else if (drawingID == CIRCLE) {

drawingData.circleData.push(drawingData.redoDrawing.pop());
                drawingData.undoDrawingOrder.push(CIRCLE); //Pushing a
CIRCLE means the circle tool was used, so if we undo any drawings, we
undo from the circle array
                var circle = new createjs.Shape();
                circle.name = Math.floor(Date.now() * Math.random());
                circle.id = CIRCLE;
                circle.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
                circle.graphics.setStrokeStyle(1);

circle.graphics.beginStroke(drawingData.circleData[drawingData.circleDa
ta.length - 1].color)

circle.graphics.beginFill(hex2rgba(drawingData.circleData[drawingData.c
ircleData.length - 1].color,
drawingData.circleData[drawingData.circleData.length - 1].fillColor));

circle.graphics.drawCircle(drawingData.circleData[drawingData.circleDat
a.length - 1].x * ratio,
drawingData.circleData[drawingData.circleData.length - 1].y * ratio,
drawingData.circleData[drawingData.circleData.length - 1].radius *
ratio);
                circle.graphics.endStroke();
                drawingData.circleData[drawingData.circleData.length -
1].circle = circle;

stage.addChild(drawingData.circleData[drawingData.circleData.length -
1].circle);
                stage.update();
                lastLabelCheck(false, CIRCLE); //False because that
will trigger the redo side of the function (add a label)
            }
            else if (drawingID == AUTO_CLUSTER) //Won't trigger till
tool is implemented
            {

drawingData.auto_clusterData.push(drawingData.redoDrawing.pop());
                drawingData.undoDrawingOrder.push(AUTO_CLUSTER);
//Pushing a AUTO_CLUSTER means the auto_cluster tool was used, so if we
undo any drawings, we undo from the auto_cluster array
                lastLabelCheck(false, AUTO_CLUSTER); //False because
that will trigger the redo side of the function (add a label)
            }
            else if (drawingID == FLOOD_FILL) //Won't trigger till tool
is implemented
```

```
            {

drawingData.flood_fillData.push(drawingData.redoDrawing.pop());
                drawingData.undoDrawingOrder.push(FLOOD_FILL);
//Pushing a FLOOD_FILL means the flood_fill was used, so if we undo any
drawings, we undo from the flood_fill array
                lastLabelCheck(false, FLOOD_FILL); //False because that
will trigger the redo side of the function (add a label)
            }
            else //Should never hit, but just incase something weird
happens
            {
                alertify.error("We could not find anything to redo.");
            }
        }
        else {
            alertify.error("You must undo something in order to redo
it.");
        }
    }
}

/****Code taken from http://wowmotty.blogspot.com/2009/06/convert-
jquery-rgb-output-to-hex-color.html *******************/
function rgb2hex(orig){ //Prevents the CSV file from removing the
commas that would be in the rgb by converting rgb to hex
 var rgb = orig.replace(/\s/g,'').match(/^rgba?\((\d+),(\d+),(\d+)/i);
 return (rgb && rgb.length === 4) ? "#" +
  ("0" + parseInt(rgb[1],10).toString(16)).slice(-2) +
  ("0" + parseInt(rgb[2],10).toString(16)).slice(-2) +
  ("0" + parseInt(rgb[3],10).toString(16)).slice(-2) : orig;
}
/***********************************************************************
*****************************************************/

/*******Code lightly modified from
http://jsfiddle.net/subodhghulaxe/t568u/ *******/
function hex2rgba(hex, opacity)
{
    hex = hex.replace('#', '');
    var r = parseInt(hex.substring(0, 2), 16);
    var g = parseInt(hex.substring(2, 4), 16);
    var b = parseInt(hex.substring(4, 6), 16);

    var result = 'rgba(' + r + ',' + g + ',' + b + ',' + opacity + ')';
    return result;
}
/***********************************************************************
************/

function clearStageForNewImage()
{
    stage.removeAllChildren(); //Clears everything on the stage to make
room for the new stage
    myGraphics.removeAllChildren; //remove all shape objects
    myGraphics.clear() //Wipe away everything so we can redraw it at
the correct scale.
```

```
    drawingData.wipeAll(); //Empty out all the arrays that help our
drawing information
    turnOffButton(); //Make it so no buttons are selected
    stage.update();

    //Remove all event listeners

document.getElementById("myCanvas").removeEventListener("mouseover",
displayCoordinates); //Displays mouse coordinates when over canvas

document.getElementById("myCanvas").removeEventListener("mouseover",
zoomResize); //Zooms in and out the image by scrolling

document.getElementById("myCanvas").removeEventListener("mouseover",
zoomDrag); //Allows the user to move the image around after its been
zoomed in on

document.getElementById("myCanvas").removeEventListener("mouseover",
scrollResize); //scrolling over the image
    stage.removeEventListener("click", grabShapeObject); //scrolling
over the image
    $("#scaledImage").off("keydown"); //shift scroll - needs testing
    document.getElementById("pointBtn").removeEventListener("click",
drawPoint); //draw point
    document.getElementById("pencilBtn").removeEventListener("click",
drawPencil); //draw pencil
    document.getElementById("lineBtn").removeEventListener("click",
drawLine); //draw line
    document.getElementById("polylineBtn").removeEventListener("click",
drawPolyline); //draw polyline
    document.getElementById("polygonBtn").removeEventListener("click",
drawPolygon); //draw polygon
    document.getElementById("circleBtn").removeEventListener("click",
drawCircle); //draw circle

document.getElementById("auto_clusterBtn").removeEventListener("click",
drawAuto_Cluster); //draw auto_cluster

document.getElementById("flood_fillBtn").removeEventListener("click",
drawFlood_Fill); //draw flood_fill
    document.getElementById("save").removeEventListener('click', save);
//Save
    document.getElementById("undo").removeEventListener('click', undo);
//Undo
    document.getElementById("redo").removeEventListener('click', redo);
//redo
    document.getElementById("delete-btn").removeEventListener('click',
deleteDrawing); //delete drawing
    document.getElementById("load-btn").removeEventListener('click',
loadDrawingButton); //load in a previous drawing to made edits
    document.getElementById("size").removeEventListener("click",
resize); //draw after changing the slider
    document.getElementById('upload').removeEventListener('change',
drawImage);
    document.getElementById("tutorial2").removeEventListener('click',
tutorial); //tutorial for app
```

```
    document.getElementById("btnLabel").removeEventListener("click",
label); //Label
    document.getElementById("labelForm").removeEventListener("submit",
label); //listens for enter key
    document.getElementById("scaleOut").removeEventListener("click",
scaleOutButton); //Scale out button
    document.getElementById("scaleIn").removeEventListener("click",
scaleInButton); //Scale in button
    document.getElementById("dropDown").removeEventListener("change",
displayShapeByLabel); //drop down menu -> drawing only labels selected

/******************************Colors*********************************
*******/
    document.getElementById("btn-black").removeEventListener("click",
color.black); //Black
    document.getElementById("btn-white").removeEventListener("click",
color.white); //White
    document.getElementById("btn-darkRed").removeEventListener("click",
color.darkRed); //Dark Red
    document.getElementById("btn-red").removeEventListener("click",
color.red); //Red
    document.getElementById("btn-orange").removeEventListener("click",
color.orange); //Orange
    document.getElementById("btn-yellow").removeEventListener("click",
color.yellow); //Yellow
    document.getElementById("btn-
neonGreen").removeEventListener("click", color.neonGreen); //Neon Green
    document.getElementById("btn-
pukeGreen").removeEventListener("click", color.pukeGreen); //Puke Green
    document.getElementById("btn-
lightBlue").removeEventListener("click", color.lightBlue); //Light Blue
    document.getElementById("btn-blue").removeEventListener("click",
color.blue); //Blue
    document.getElementById("btn-purple").removeEventListener("click",
color.purple); //Purple
    document.getElementById("btn-pink").removeEventListener("click",
color.pink); //Pink
    document.getElementById("btn-user1").removeEventListener("click",
color.user1); //User1 (User ones are colors that the user can pick from
a color wheel (not implemented yet)
    document.getElementById("btn-user2").removeEventListener("click",
color.user2); //User2
    document.getElementById("btn-user3").removeEventListener("click",
color.user3); //User3
    document.getElementById("btn-user4").removeEventListener("click",
color.user4); //User4

document.getElementById("colorPickerBtn").removeEventListener("click",
assignColor); //color wheel button assigns color to a color.user button

document.getElementById("colorWheelForm").removeEventListener("submit",
colorWheelText); //listens for enter key

/*********************************************************************
*******/

    /*Wipe our old labels away from the drop down menu*/
    var dropDownMenu = document.getElementById("dropDown");
    for (var i = dropDownMenu.length; i > 0; i--) //Cycle through the
dropdown menu backwards, ignoring the first one ("All Labels")
    {
        dropDownMenu.remove(i); //Remove the label from the drop down
menu
    }
    /***************************************************/
}

/****Deals with drawing and drag scrolling at the same time****/
function setCursorForDrawing() //Prevents user from drag scrolling
while a tool is selected and provides a visual for this
{
    if ($('#scaledImage').hasClass('dragscroll')) //If we can drag
scroll
    {
        $('#scaledImage').removeClass('dragscroll');
        dragscroll.reset(); //Updates dragscroll listener
        $('#myCanvas').removeClass('grabCursor');
        $('#myCanvas').addClass('crosshairCursor');
    }
}
function setCursorForDragging() //Allows the user to drag scroll and
provides a visual for this
{
    if (!$('#scaledImage').hasClass('dragscroll')) //If we don't have
the dragscroll as a class
    {
        $('#scaledImage').addClass('dragscroll');
        dragscroll.reset(); //Updates dragscroll listener
        $('#myCanvas').addClass('grabCursor');
        $('#myCanvas').removeClass('crosshairCursor');
    }
}
function currentlyDrawing()
{
    if ($('#scaledImage').hasClass('dragscroll')) //If we can drag
scroll then we aren't currently drawing
    {
        return false;
    }
    else //We have a tool selected and thus are drawing
    {
        return true;
    }
}
/*********************************************************/

function deleteDrawing()
{
    if (drawingData.shapeToDelete != null) //Make sure something has
been selected on the stage, even if just the bitmap
    {
        if (drawingData.shapeToDelete.name != null &&
drawingData.shapeToDelete.name != "image") //Makes sure a shape is
selected
```

```
        {
            alertify.confirm("Are you sure you want to delete this
shape? You cannot undo this action.", function (e)
            {
                if (e) //If they said yes
                {
                    removeTooltip();//Remove tooltip -> no need to
deselect shape as we are deleting it anyways
                    var positionInArray = 0;
                    if (drawingData.shapeToDelete.id == POINT)
//Deleting a point
                    {
                        for (var i = 0; i <
drawingData.pointData.length; i++) //Find the correct name
                        {
                            if (drawingData.shapeToDelete.name ==
drawingData.pointData[i].point.name) {
                                positionInArray = i;

stage.removeChild(drawingData.pointData[i].point); //Remove it from the
stage
                                stage.update();
                                removeShapeFromUndoArray(POINT, i);
                                break; //Get out of the loop
                            }
                        }
                        drawingData.pointData.splice(positionInArray,
1); //Remove the shape from our array and then update our array
                        //Undo/Redo array pushes go here if I decide to
allow a user to undo their delete (which I probably should, but
currently don't)
                    }
                    else if (drawingData.shapeToDelete.id == PENCIL)
//Deleting a pencil
                    {
                        for (var i = 0; i <
drawingData.pencilData.length; i++) //Find the correct name
                        {
                            if (drawingData.shapeToDelete.name ==
drawingData.pencilData[i].pencil.name) {
                                positionInArray = i;

stage.removeChild(drawingData.pencilData[i].pencil); //Remove it from
the stage
                                stage.update();
                                removeShapeFromUndoArray(PENCIL, i);
                                break; //Get out of the loop
                            }
                        }
                        drawingData.pencilData.splice(positionInArray,
1); //Remove the shape from our array and then update our array
                        //Undo/Redo array pushes go here if I decide to
allow a user to undo their delete (which I probably should, but
currently don't)
                    }
                    else if (drawingData.shapeToDelete.id == LINE)
//Deleting a line
```

```
                    {
                        for (var i = 0; i <
drawingData.lineData.length; i++) //Find the correct name
                        {
                            if (drawingData.shapeToDelete.name ==
drawingData.lineData[i].lineObject.name) {
                                positionInArray = i;

stage.removeChild(drawingData.lineData[i].lineObject); //Remove it from
the stage
                                stage.update();
                                removeShapeFromUndoArray(LINE, i);
                                break; //Get out of the loop
                            }
                        }
                        drawingData.lineData.splice(positionInArray,
1); //Remove the shape from our array and then update our array
                        //Undo/Redo array pushes go here if I decide to
allow a user to undo their delete (which I probably should, but
currently don't)
                    }
                    else if (drawingData.shapeToDelete.id == POLYLINE)
//Deleting a polyline
                    {
                        for (var i = 0; i <
drawingData.polylineData.length; i++) //Find the correct name
                        {
                            if (drawingData.shapeToDelete.name ==
drawingData.polylineData[i].polyline.name) {
                                positionInArray = i;

stage.removeChild(drawingData.polylineData[i].polyline); //Remove it
from the stage
                                stage.update();
                                removeShapeFromUndoArray(POLYLINE, i);
                                break; //Get out of the loop
                            }
                        }

drawingData.polylineData.splice(positionInArray, 1); //Remove the shape
from our array and then update our array
                        //Undo/Redo array pushes go here if I decide to
allow a user to undo their delete (which I probably should, but
currently don't)
                    }
                    else if (drawingData.shapeToDelete.id == POLYGON)
//Deleting a polygon
                    {
                        for (var i = 0; i <
drawingData.polygonData.length; i++) //Find the correct name
                        {
                            if (drawingData.shapeToDelete.name ==
drawingData.polygonData[i].polygon.name) {
                                positionInArray = i;

stage.removeChild(drawingData.polygonData[i].polygon); //Remove it from
the stage
```

```
                                    stage.update();
                                    removeShapeFromUndoArray(POLYGON, i);
                                    break; //Get out of the loop
                                }
                            }
                            drawingData.polygonData.splice(positionInArray,
1); //Remove the shape from our array and then update our array
                            //Undo/Redo array pushes go here if I decide to
allow a user to undo their delete (which I probably should, but
currently don't)
                        }
                        else if (drawingData.shapeToDelete.id == CIRCLE)
//Deleting a circle
                        {
                            for (var i = 0; i <
drawingData.circleData.length; i++) //Find the correct name
                            {
                                if (drawingData.shapeToDelete.name ==
drawingData.circleData[i].circle.name) {
                                    positionInArray = i;

stage.removeChild(drawingData.circleData[i].circle); //Remove it from
the stage
                                    stage.update();
                                    removeShapeFromUndoArray(CIRCLE, i);
                                    break; //Get out of the loop
                                }
                            }
                            drawingData.circleData.splice(positionInArray,
1); //Remove the shape from our array and then update our array
                            //Undo/Redo array pushes go here if I decide to
allow a user to undo their delete (which I probably should, but
currently don't)
                        }
                        else if (drawingData.shapeToDelete.id ==
AUTO_CLUSTER) //Deleting an auto_cluster
                        {
                            //Nothing to do yet
                        }
                        else if (drawingData.shapeToDelete.id ==
FLOOD_FILL) //Deleting a flood_fill
                        {
                            //Nothing to do yet
                        }
                        else //id exists but does not match any drawing
                        {
                            alertify.error("We could not find this object
to delete. Please try again or refresh the page.");
                        }
                        drawingData.shapeToDelete = null; //Clear object
                        alertify.success("Shape deleted successfully.");
                    }
                    else {
                        //Do nothing (User clicked cancel)
                    }

                });
```

```
        }
        else //If we didn't find the shape to delete
        {
            alertify.error("No shape is selected. Please select a shape
and try again.");
        }
    }
    else //User most likely selected the image and not a shape
    {
        alertify.error("No shape is selected. Please select a shape and
try again.");
    }

}

function loadDrawingButton()
{
    alertify.confirm("Are you sure you want to load in previously drawn
data to this image? This will permanently delete all currently drawn
shapes.", function (e)
    {
        if (e)
        {
            //I delete all current drawings so we don't have to check
collision on all loaded in drawings.
            //This will decrease load in time as I won't have to check
each new shape against all other shapes and thus prevent dupilicate
date
            wipeDrawingsForLoad(); //remove all currently drawn data so
we don't have to worry about overlaping drawings (or user loading in
more than 1 drawing data set)
            document.getElementById('load').addEventListener('change',
loadDrawingData);
            if (window.File && window.FileReader && window.FileList &&
window.Blob) {
                $("#load").trigger('click');
            }
            else {
                switchAlertLabelForAlert();
                alertify.alert("We're sorry, but the File APIs are not
fully supported in this browser Please try a different browser.");
                switchAlertLabelForConfirm();
            }
        }
        else
        {
            //Do nothing -> its as if they never hit the load button in
the first place
        }
    });
}

function loadDrawingData(e)
{
    /*************************Code for reading a csv file was modified
from these two websites***************************/
```

```
    //http://www.htmlgoodies.com/beyond/javascript/read-text-files-
using-the-javascript-filereader.html#fbid=vuei5iprIXD /
    ///http://stackoverflow.com/questions/23331546/how-to-use-
javascript-to-read-local-text-file-and-read-line-by-line ///

/***********************************************************************
***********************************************/
    var drawingDataFile = e.target.files[0];     //Retrieve the first
(and only!) File from the FileList object

    if (!drawingDataFile)
    {
        switchAlertLabelForAlert();
        alertify.alert("Failed to load file");
        switchAlertLabelForConfirm();
    }
    else if (drawingDataFile.type.match('text/csv.*') ||
drawingDataFile.type.match('text/comma-seperated-values.*') ||
drawingDataFile.type.match('application/vnd.ms-excel.*')
            || drawingDataFile.type.match('application/csv.*') ||
drawingDataFile.type.match('application/excel.*') ||
drawingDataFile.type.match('application/vnd.msexcel.*')) //If the file
is a csv file
    {
        imageName = imageName.replace(/\.[^/.]+$/, ""); //Clean image
file name so we can compare to see if this image lines up with the
drawing data
        var imageNameInFile = drawingDataFile.name.substring(0,
drawingDataFile.name.indexOf('_')); //Because of this line here, we
changed imageName to not have any "_" in it, otherwise it would stop
short
        var imageAndFileMatch = false;
        if (imageNameInFile === imageName) //If the image name is the
same as the file name
        {
            imageAndFileMatch = true; //Can open file
        }
        if (!imageAndFileMatch) {
            switchAlertLabelForAlert();
            alertify.alert("Error: The image currently selected and the
file you selected to load in do not match and therefore we cannot load
in the drawing data you have selected. If you believe this to be a
mistake, please double check your file names.", function () {
                alertify.log("Load Canceled.", "error", 10000); //Will
be displayed after user clicks ok on the alert above
            });
            switchAlertLabelForConfirm();
        }
        else {
            //Resize image to original image size
            //Just resizing the image to the original size on load is
faster than resizing every single drawing in our load file to whatever
size the user currently has the image at
            if (potraitLayout) {
                document.getElementById("size").value = originalHeight;
//Set slider bar to correct size
            }
            else {
                document.getElementById("size").value = originalWidth;
//Set slider bar to correct size
            }
            if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
            {
                stage.canvas.width = originalWidth; //set canvas to
correct size
            }
            if (stage.canvas.height != originalHeight) {
                stage.canvas.height = originalHeight; //set canvas to
correct size
            }
            resize(); //Now that canvas size is reset, reset image size
            document.getElementById("spinner").style.visibility =
"visible";
            var drawingInfo = new FileReader();
            drawingInfo.onload = function (event)
            {
                var contents = this.result.split('\n');
                for (var line = 1; line < contents.length - 1; line++)
// start at 1 because we don't care how many shapes there are (first
entry in file is number of shapes)
                {
                    if (!drawLoadedDrawings(contents[line])) //If load
data contained bad data
                    {
                        if (drawingData.pointData.length > 0 ||
drawingData.pencilData.length > 0 || drawingData.lineData.length > 0 ||
drawingData.polylineData.length > 0
                           || drawingData.polygonData.length > 0 ||
drawingData.circleData.length > 0 ||
drawingData.auto_clusterData.length > 0 ||
drawingData.flood_fillData.length > 0) //if we have loaded anything
before fail
                        {
                            wipeDrawingsForLoad(); //Ultimately, load
failed so we delete everything and start fresh. Could remove this if
statement and have data loaded up to the error point if desired.
                        }
                        return; //Get out - bad data in load file
                    }
                }
                stage.update(); //Show all the drawings we just loaded
in on the screen
                document.getElementById("spinner").style.visibility =
"hidden"; //Hide the spinner to let user know we're done loading
                alertify.log("Warning: If the data you loaded in was
not created with the specific image you are currently looking at then
your data may be inaccurate.", "warning", 15000); //Do this always, or
if image name != current image name???

            };
            drawingInfo.readAsText(drawingDataFile);
        }
    }
    else //Not a csv file
    {
        switchAlertLabelForAlert();
        alertify.alert(drawingDataFile.name + " is not a valid csv
file.");
        switchAlertLabelForConfirm();
    }

}

function tutorial()
{
    switchAlertLabelForAlert();
    //Spilt up tutorial text so it is easier to read for the
programmer.
    var tutorialText = "Welcome to the FireMAP Training Data Selector.
Our classifier needs to know what's what to accurately classify your
images.";
    tutorialText += " In order to do this, upload an image that is an
accurate representation of the data you will be classifying.";
    tutorialText += " Once it is uploaded you can then draw on the
image using any tool from the Tool Selector Menu.";
    tutorialText += "<br/><br/>After you have drawn a shape on your
image you can then label the shape using the label textbox and
button.";
    tutorialText += " You can see what individual shapes have been
labeled by clicking on them and you can see every shape that has the
same label by clicking on the \"Showing\" drop down menu.";
    tutorialText += " Once a shape is labeled it cannot be relabeled,
but it can be deleted with our delete tool.";
    tutorialText += " Shapes can also be undone and redone if you make
a mistake while drawing the shape.<br/><br/>The Color Picker menu
allows you to select what color the shape will be drawn in.";
    tutorialText += " This allows the shape to stand out better against
your training image.";
    tutorialText += " If one of the default colors does not work well
for you, you can always select your own color via the color wheel and
then hitting the \"Assign Color\" button";
    tutorialText += " or by entering your own hex value into the color
wheel textbox.";
    tutorialText += " The color you select does not affect the results
of the classifier.<br/><br/>To extract the labels and coordinates of
your drawings, simply hit the \"Save\" button";
    tutorialText += " and a download option will pop up shortly. If you
have a previously saved training data file you can load it in via the
\"Load\" button if the image currently showing";
    tutorialText += " is the same image that the data was originally
drawn on.<br/><br/>This application does allow the user to zoom in on
the image as well as rescale the image.";
    alertify.alert(tutorialText);
    switchAlertLabelForConfirm();
}

function grabShapeObject(e)
{
    var drawing = currentlyDrawing();
    if (!drawing) //If aren't drawing, then we can drag. If we are, we
have to unselect the tool to keep dragging
    {
        drawingData.shapeToDelete = e.target;
        unSelectedObject();//unselect currently selected object (if one
is selected)
        if (e.target.id == POINT) //If shape we clicked on was a
point...
        {
            for (var i = 0; i < drawingData.pointData.length; i++)
            {
                if (drawingData.pointData[i].point.name ==
e.target.name) //Find the point in our array of points that we've
clicked on so we can "highlight" it to the user, showing them they've
selected it
                {

drawingData.pointData[i].point.shadow.setShadow("#ffff00", 0, 0, 7);
//Creates a highlighe affect on the shape to show that it is currently
being selected
                    drawingData.selectedObject[0] = POINT;
                    drawingData.selectedObject[1] = i;
                    //Add label tooltip where mouse was clicked
                    addTooltip(i, drawingData.pointData);
                    break; //We've found the shape, so we can stop
processing the loop as we are only changing this one shape
                }
            }
        }
        else if (e.target.id == PENCIL)//If shape we clicked on was a
pencil drawing...
        {
            for (var i = 0; i < drawingData.pencilData.length; i++) {
                if (drawingData.pencilData[i].pencil.name ==
e.target.name) //Find the point in our array of points that we've
clicked on so we can "highlight" it to the user, showing them they've
selected it
                {

drawingData.pencilData[i].pencil.shadow.setShadow("#ffff00", 0, 0,
7);//Creates a highlighe affect on the shape to show that it is
currently being selected
                    drawingData.selectedObject[0] = PENCIL;
                    drawingData.selectedObject[1] = i;
                    //Add label tooltip where mouse was clicked
                    addTooltip(i, drawingData.pencilData);
                    break; //We've found the shape, so we can stop
processing the loop as we are only changing this one shape
                }
            }
        }
        else if (e.target.id == LINE)//If shape we clicked on was a
line...
        {
            for (var i = 0; i < drawingData.lineData.length; i++) {
                if (drawingData.lineData[i].lineObject.name ==
e.target.name) //Find the point in our array of points that we've
```

```
clicked on so we can "highlight" it to the user, showing them they've
selected it
                {

drawingData.lineData[i].lineObject.shadow.setShadow("#ffff00", 0, 0,
7);//Creates a highlighe affect on the shape to show that it is
currently being selected
                    drawingData.selectedObject[0] = LINE;
                    drawingData.selectedObject[1] = i;
                    //Add label tooltip where mouse was clicked
                    addTooltip(i, drawingData.lineData);
                    break; //We've found the shape, so we can stop
processing the loop as we are only changing this one shape
                }
            }
        }
        else if (e.target.id == POLYLINE)//If shape we clicked on was a
polyline...
        {
            for (var i = 0; i < drawingData.polylineData.length; i++) {
                if (drawingData.polylineData[i].polyline.name ==
e.target.name) //Find the point in our array of points that we've
clicked on so we can "highlight" it to the user, showing them they've
selected it
                {

drawingData.polylineData[i].polyline.shadow.setShadow("#ffff00", 0, 0,
7);//Creates a highlighe affect on the shape to show that it is
currently being selected
                    drawingData.selectedObject[0] = POLYLINE;
                    drawingData.selectedObject[1] = i;
                    //Add label tooltip where mouse was clicked
                    addTooltip(i, drawingData.polylineData);
                    break; //We've found the shape, so we can stop
processing the loop as we are only changing this one shape
                }
            }
        }
        else if (e.target.id == POLYGON)//If shape we clicked on was a
polygon...
        {
            for (var i = 0; i < drawingData.polygonData.length; i++) {
                if (drawingData.polygonData[i].polygon.name ==
e.target.name) //Find the point in our array of points that we've
clicked on so we can "highlight" it to the user, showing them they've
selected it
                {

drawingData.polygonData[i].polygon.shadow.setShadow("#ffff00", 0, 0,
7);//Creates a highlighe affect on the shape to show that it is
currently being selected
                    drawingData.selectedObject[0] = POLYGON;
                    drawingData.selectedObject[1] = i;
                    //Add label tooltip where mouse was clicked
                    addTooltip(i, drawingData.polygonData);
                    break; //We've found the shape, so we can stop
processing the loop as we are only changing this one shape
```

```
                }
            }
        }
        else if (e.target.id == CIRCLE)//If shape we clicked on was a
circle...
        {
            for (var i = 0; i < drawingData.circleData.length; i++) {
                if (drawingData.circleData[i].circle.name ==
e.target.name) //Find the point in our array of points that we've
clicked on so we can "highlight" it to the user, showing them they've
selected it
                {

drawingData.circleData[i].circle.shadow.setShadow("#ffff00", 0, 0, 7);
                    drawingData.selectedObject[0] = CIRCLE;
                    drawingData.selectedObject[1] = i;
                    //Add label tooltip where mouse was clicked
                    addTooltip(i, drawingData.circleData);
                    break; //We've found the shape, so we can stop
processing the loop as we are only changing this one shape
                }
            }
        }
        else if (e.target.id == AUTO_CLUSTER)//If shape we clicked on
was an auto-cluster object...
        {
            alert("This feature is not yet implemented");
        }
        else if (e.target.id == FLOOD_FILL)//If shape we clicked on was
a flood_fill object...
        {
            alert("This feature is not yet implemented");
        }
        stage.update();
    }
}

/*Check to see if our undo or delete was the last label one from the
dropdown menu. If so, remove from the menu*/
function lastLabelCheck(undo, id)
{
    var matchFound = 0;
    if (id == POINT)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if (drawingData.pointData[drawingData.pointData.length
- 1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if (drawingData.pointData[drawingData.pointData.length
- 1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == PENCIL)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.pencilData[drawingData.pencilData.length - 1].label ===
drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++)
            {
                if
(drawingData.pencilData[drawingData.pencilData.length - 1].label ===
drawingData.labelCount[i])
                {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == LINE)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if (drawingData.lineData[drawingData.lineData.length -
1].label === drawingData.labelCount[i]) {
```

```
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if (drawingData.lineData[drawingData.lineData.length -
1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == POLYLINE)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.polylineData[drawingData.polylineData.length - 1].label
=== drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.polylineData[drawingData.polylineData.length - 1].label
=== drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == POLYGON)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.polygonData[drawingData.polygonData.length - 1].label ===
drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.polygonData[drawingData.polygonData.length - 1].label ===
drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == CIRCLE)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.circleData[drawingData.circleData.length - 1].label ===
drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.circleData[drawingData.circleData.length - 1].label ===
drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == AUTO_CLUSTER)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.auto_clusterData[drawingData.auto_clusterData.length -
1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.auto_clusterData[drawingData.auto_clusterData.length -
1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
    else if (id == FLOOD_FILL)
    {
        if (undo) //Subtract one from our label count for that
particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.flood_fillData[drawingData.flood_fillData.length -
1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] - 1; //Subtract one from the count of
this label
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
        else //Add one to our label count for that particular label
        {
            for (var i = 0; i < drawingData.labelCount.length; i++) {
                if
(drawingData.flood_fillData[drawingData.flood_fillData.length -
1].label === drawingData.labelCount[i]) {
                    drawingData.labelCount[i + 1] =
drawingData.labelCount[i + 1] + 1; //Add one to the count of this label
```

```
                    break; //No need to go through the rest of the for
loop as we found what we need
                }
            }
        }
    }
}
/***********************************************************************
*****************************************/

function amountOfDifferentLabels()
{
    var labelCount = 0;
    for (var i = 1; i < drawingData.labelCount.length;)
    {
        if (drawingData.labelCount[i] > 0) //If our label has a count
associated with it
        {
            labelCount++;
        }
        else
        {
            var unusedLabel = drawingData.labelCount[i - 1];
            drawingData.unusedLabels.push(unusedLabel); //Push the
label that doesn't have a count so we can remove it from our list on
save
        }
        i += 2; //Every second spot in the array starting with index 1
is a number, not comparing a strings so we can skip those
    }
    return labelCount;
}

function drawCompletePolygon(drawLastPolygon, ratio)
{
    var loadedInPolygon = false; //Used only if user loads in a file
containing polygons
    if (ratio == -1) //loaded in polyline sends -1, any other function
call would be a positive number
    {
        loadedInPolygon = true;
        ratio = null; //null so we'll do a correct calculation of size
later
    }
    else
    {
        loadedInPolygon = false;
    }
    if (ratio == null)
    {
        ratio;
        var size = document.getElementById("size").value;
        if (potraitLayout) {
            ratio = size / originalHeight;
        }
        else {
            ratio = size / originalWidth;
        }
    }
    var newPolygon = false; //Only draw if something needs to be
drawn/redrawn.
    if (drawLastPolygon) //if only drawing the last one (no need to
redraw every polygon when we make 1 new one)
    {
        var polygon = new createjs.Shape();
        polygon.id = POLYGON;
        polygon.shadow = new createjs.Shadow("#000", 0, 0, 0); //Create
a new shadow that will display if the user selects this shape after
creating it to show the user that this shape is currently selected
        var numberOfChildren = stage.getNumChildren();
        if (loadedInPolygon) //Allows us to skip the for loop below if
we have loaded in a polyline -> increases speed up
        {
            numberOfChildren = 0;
        }
        for (var i = numberOfChildren - 1; i > 0; i--)
        {
            var child = stage.getChildAt(i);
            if (UNCOMPLETED_POLYGON == child.name) //if an uncompleted
polygon line segment is found, remove it and say we have a new polygon
to be drawn
            {
                stage.removeChildAt(i);
                newPolygon = true;
            }
        }
        if (newPolygon || loadedInPolygon)
        {
            var lineSegmentCounter = 0;

polygon.graphics.beginStroke(drawingData.polygonData[drawingData.polygo
nData.length - 1].color);

polygon.graphics.beginFill(hex2rgba(drawingData.polygonData[drawingData
.polygonData.length - 1].color,
drawingData.polygonData[drawingData.polygonData.length -
1].fillColor));

polygon.graphics.moveTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
            //Loop through all the points for this polygon and draw
them
            for (var i = 1; i <
drawingData.polygonData[drawingData.polygonData.length - 1].x.length;
i++) {

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[i] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[i] *
ratio)); //Draw to the next point on the polygon
                lineSegmentCounter++;
            }
            if (loadedInPolygon) //When we load in a line segment we do
it as 1 whole "line" thus, no segments
            {
                lineSegmentCounter = 0;
            }

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Connect last point with the beginning point
            polygon.graphics.endStroke();
            polygon.name = "completedPolygon" + Math.floor(Date.now() *
Math.random());
            drawingData.polygonData[drawingData.polygonData.length -
1].polygon = polygon;

stage.addChild(drawingData.polygonData[drawingData.polygonData.length -
1].polygon);
            if (!loadedInPolygon) //Only reset length if actual new
polyline
            {
                drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
            }
            for (var i = 1; i < lineSegmentCounter; i++) //Will take
all but 1 off the polygon undo stack. The 1 there will represent the
whole polygon and not just a line segment
            {
                drawingData.undoDrawingOrder.pop();
            }
        }
    }
    else //redrawing all polygons
    {
        var numberOfChildren = stage.getNumChildren();
        for (var i = numberOfChildren - 1; i > 0; i--) {
            var child = stage.getChildAt(i);
            if (UNCOMPLETED_POLYGON == child.name) //if an uncompleted
polygon line segment is found, remove it and say we have a new polygon
to be drawn
            {
                stage.removeChildAt(i);
                newPolygon = true;
            }
        }
        if (newPolygon) //if I am making a new polygon
        {
            var polygon = new createjs.Shape();
            polygon.id = POLYGON;
            polygon.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            var lineSegmentCounter = 0;
```

```
polygon.graphics.beginStroke(drawingData.polygonData[drawingData.polygo
nData.length - 1].color);

polygon.graphics.beginFill(hex2rgba(drawingData.polygonData[drawingData
.polygonData.length - 1].color,
drawingData.polygonData[drawingData.polygonData.length -
1].fillColor));

polygon.graphics.moveTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
            //Loop through all the points for this polygon and draw
them
            for (var i = 1; i <
drawingData.polygonData[drawingData.polygonData.length - 1].x.length;
i++) {

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[i] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[i] *
ratio)); //Draw to the next point on the polygon
                lineSegmentCounter++;
            }

polygon.graphics.lineTo((drawingData.polygonData[drawingData.polygonDat
a.length - 1].x[0] * ratio),
(drawingData.polygonData[drawingData.polygonData.length - 1].y[0] *
ratio)); //Connect last point with the beginning point
            polygon.graphics.endStroke();
            polygon.name = "completedPolygon" + Math.floor(Date.now() *
Math.random());
            drawingData.polygonData[drawingData.polygonData.length -
1].polygon = polygon;

stage.addChild(drawingData.polygonData[drawingData.polygonData.length -
1].polygon);
            drawingData.redoDrawing.length = 0; //Empty my redo array
as I just added something - Will need to do this everytime I draw
something
            for (var i = 1; i < lineSegmentCounter; i++) //Will take
all but 1 off the polygon undo stack. The 1 there will represent the
whole polygon and not just a line segment
            {
                drawingData.undoDrawingOrder.pop();
            }
        }
        var polygonDataLength;
        if (!newPolygon) //All polygons are completed, therefore,
redraw all
        {
            polygonDataLength = drawingData.polygonData.length;
        }
        else
        {
            polygonDataLength = drawingData.polygonData.length - 1;
//if we've already drawn the last polygon added (because we auto-
completed) no need to redraw it
        }
        for (var i = 0; i < polygonDataLength; i++) //-1 because
        {
            var polygon = new createjs.Shape();
            polygon.id = POLYGON;
            polygon.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected

polygon.graphics.beginStroke(drawingData.polygonData[i].color);

polygon.graphics.beginFill(hex2rgba(drawingData.polygonData[i].color,
drawingData.polygonData[i].fillColor));
            polygon.graphics.moveTo((drawingData.polygonData[i].x[0] *
ratio), (drawingData.polygonData[i].y[0] * ratio)); //Move to the
starting position of the first line
            for (var j = 0; j < drawingData.polygonData[i].x.length;
j++)
            {

polygon.graphics.lineTo((drawingData.polygonData[i].x[j] * ratio),
(drawingData.polygonData[i].y[j] * ratio)); //Draw to the next point on
the polygon
            }
            polygon.graphics.lineTo((drawingData.polygonData[i].x[0] *
ratio), (drawingData.polygonData[i].y[0] * ratio)); //Connect last
point with the beginning point
            polygon.graphics.endStroke();
            polygon.name = "completedPolygon" + Math.floor(Date.now() *
Math.random());
            drawingData.polygonData[i].polygon = polygon;
            stage.addChild(drawingData.polygonData[i].polygon);
        }
        polyButtons.polyFirstTime = true; //Set flag showing that the
next time we draw a polygon, it will be a new one as redrawing
completes any uncompleted polygons
    }
    stage.update();
}

function drawCompletePolyline(drawLastPolyline, ratio)
{
    var loadedInPolyline = false; //Used only if user loads in a file
containing polylines
    if (ratio == -1) //loaded in polyline sends -1, any other function
call would be a positive number
    {
        loadedInPolyline = true;
        ratio = null; //null so we'll do a correct calculation of size
later
    }
    else
    {
        loadedInPolyline = false;
    }
    if (ratio == null) {
        ratio;
        var size = document.getElementById("size").value;
        if (potraitLayout) {
            ratio = size / originalHeight;
        }
        else {
            ratio = size / originalWidth;
        }
    }
    var newPolyline = false; //Only draw if something needs to be
drawn/redrawn.
    if (drawLastPolyline) //if only drawing the last one (no need to
redraw every polyline when we make 1 new one)
    {
        var polyline = new createjs.Shape();
        polyline.id = POLYLINE;
        polyline.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
        var numberOfChildren = stage.getNumChildren();
        if (loadedInPolyline) //Allows us to skip the for loop below if
we have loaded in a polyline -> increases speed up
        {
            numberOfChildren = 0;
        }
        for (var i = numberOfChildren - 1; i > 0; i--) {
            var child = stage.getChildAt(i);
            if (UNCOMPLETED_POLYLINE == child.name) //if an uncompleted
polyline line segment is found, remove it and say we have a new
polyline to be drawn
            {
                stage.removeChildAt(i);
                newPolyline = true;
            }
        }
        stage.update();
        if (newPolyline || loadedInPolyline) {
            var lineSegmentCounter = 0;

polyline.graphics.beginStroke(drawingData.polylineData[drawingData.poly
lineData.length - 1].color);

polyline.graphics.moveTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[0] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
            //Loop through all the points for this polyline and draw
them
            for (var i = 1; i <
drawingData.polylineData[drawingData.polylineData.length - 1].x.length;
i++) {

polyline.graphics.lineTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[i] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[i] *
ratio)); //Draw to the next point on the polyline
                lineSegmentCounter++;
            }
            if (loadedInPolyline) //When we load in a line segment we
do it as 1 whole "line" thus, no segments
            {
                lineSegmentCounter = 0;
            }
            polyline.graphics.endStroke();
            polyline.name = "completedPolyline" + Math.floor(Date.now()
* Math.random());
            drawingData.polylineData[drawingData.polylineData.length -
1].polyline = polyline;

stage.addChild(drawingData.polylineData[drawingData.polylineData.length
- 1].polyline);
            if (!loadedInPolyline) //Only reset length if actual new
polyline
            {
                drawingData.redoDrawing.length = 0; //Empty my redo
array as I just added something - Will need to do this everytime I draw
something
            }
            stage.update();
            for (var i = 1; i < lineSegmentCounter; i++) //Will take
all but 1 off the polyline undo stack. The 1 there will represent the
whole polyline and not just a line segment
            {
                drawingData.undoDrawingOrder.pop();
                stage.update();
            }
        }
    }
    else //redrawing all polylines
    {
        var numberOfChildren = stage.getNumChildren();
        for (var i = numberOfChildren - 1; i > 0; i--) {
            var child = stage.getChildAt(i);
            if (UNCOMPLETED_POLYLINE == child.name) //if an uncompleted
polyline line segment is found, remove it and say we have a new
polyline to be drawn
            {
                stage.removeChildAt(i);
                newPolyline = true;
            }
        }
        stage.update();
        if (newPolyline) //if I am making a new polyline
        {
            var polyline = new createjs.Shape();
            polyline.id = POLYLINE;
            polyline.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
```

```
            var lineSegmentCounter = 0;

polyline.graphics.beginStroke(drawingData.polylineData[drawingData.poly
lineData.length - 1].color);

polyline.graphics.moveTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[0] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[0] *
ratio)); //Move to the starting position of the first line
            //Loop through all the points for this polyline and draw
them
            for (var i = 1; i <
drawingData.polylineData[drawingData.polylineData.length - 1].x.length;
i++) {

polyline.graphics.lineTo((drawingData.polylineData[drawingData.polyline
Data.length - 1].x[i] * ratio),
(drawingData.polylineData[drawingData.polylineData.length - 1].y[i] *
ratio)); //Draw to the next point on the polyline
                lineSegmentCounter++;
            }
            polyline.graphics.endStroke();
            polyline.name = "completedPolyline" + Math.floor(Date.now()
* Math.random());
            drawingData.polylineData[drawingData.polylineData.length -
1].polyline = polyline;

stage.addChild(drawingData.polylineData[drawingData.polylineData.length
- 1].polyline);
            drawingData.redoDrawing.length = 0; //Empty my redo array
as I just added something - Will need to do this everytime I draw
something
            for (var i = 1; i < lineSegmentCounter; i++) //Will take
all but 1 off the polylines undo stack. The 1 there will represent the
whole polylines and not just a line segment
            {
                drawingData.undoDrawingOrder.pop();
            }
        }
        var polylineDataLength;
        if (!newPolyline) //All polylines are completed, therefore,
redraw all
        {
            polylineDataLength = drawingData.polylineData.length;
        }
        else {
            polylineDataLength = drawingData.polylineData.length - 1;
//if we've already drawn the last polylines added (because we auto-
completed) no need to redraw it
        }
        for (var i = 0; i < polylineDataLength; i++) //-1 because
        {
            var polyline = new createjs.Shape();
            polyline.id = POLYLINE;
            polyline.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
```

```
after creating it to show the user that this shape is currently
selected

polyline.graphics.beginStroke(drawingData.polylineData[i].color);
            polyline.graphics.moveTo((drawingData.polylineData[i].x[0]
* ratio), (drawingData.polylineData[i].y[0] * ratio)); //Move to the
starting position of the first line
            for (var j = 0; j < drawingData.polylineData[i].x.length;
j++) {

polyline.graphics.lineTo((drawingData.polylineData[i].x[j] * ratio),
(drawingData.polylineData[i].y[j] * ratio)); //Draw to the next point
on the polylines
            }
            polyline.graphics.endStroke();
            polyline.name = "completedPolyline" + Math.floor(Date.now()
* Math.random());
            drawingData.polylineData[i].polyline = polyline;
            stage.addChild(drawingData.polylineData[i].polyline);
        }
        polyButtons.polyFirstTime = true; //Set flag showing that the
next time we draw a polylines, it will be a new one as redrawing
completes any uncompleted polylines
    }
    stage.update();
}

function removeShapeFromUndoArray(shape, positionToRemove)
{
    var foundCount = -1; //Start at -1 because first find will put us
at index 0, which is correct
    for (var i = 0; i < drawingData.undoDrawingOrder.length; i++)
    {
        if (shape == drawingData.undoDrawingOrder[i]) //if id is the
same as the shape we are going to delete
        {
            foundCount++;   //Do this because we need to delete the
correct shape from our undo array and not just the same shape id
        }
        if (foundCount == positionToRemove) //Found the correct one to
delete
        {
            drawingData.undoDrawingOrder.splice(i, 1);
            break;
        }
    }
}

function keepImageInsideStageBoundaries()
{
    var localWalls = {
        topLeftCorner: stage.globalToLocal(0, 0),
        bottomRightCorner: stage.globalToLocal(stage.canvas.width,
stage.canvas.height)
    }; //Location of local walls (the boundries that you see, zoomed in
or not)
    var globalWalls = {
        topLeftCorner: stage.localToGlobal(localWalls.topLeftCorner.x,
localWalls.topLeftCorner.y),
        bottomRightCorner:
stage.localToGlobal(localWalls.bottomRightCorner.x,
localWalls.bottomRightCorner.y)
    }; //Localtion of walls on the global stage

    /*Basically when we zoom out we check the location of our local
walls and if theyre out of the global wall boundry
    then we take how much they are over and adjust the stage and the
view port based upon how much they're off. If they
    are still within the bounds then we don't do anything to them.*/
    if (localWalls.topLeftCorner.x < 0) //If overflow on the right side
(backwards I know)
        //->backwards because this should happen if there is an
overflow on the left side, not the right side, but it works the way it
should, just on the other wall
    {
        stage.x = stage.x + globalWalls.topLeftCorner.x; //Since
topLeftCorner.x is always negative, we're subtracting how far over we
are from the stage, essentially bumping our stage back on screen
        stage.regX = stage.regX - localWalls.topLeftCorner.x; //Do the
save thing for regX (essentially our viewport) except use local walls
because we set reg to local and stage to global (above code)
    }
    else if (localWalls.bottomRightCorner.x > stage.canvas.width) //If
overflow on the left side (backwards I know)
    {
        stage.x = stage.x + (globalWalls.bottomRightCorner.x -
stage.canvas.width); //Subtract the width, so we get just the amount
that we're over, then add to stage, essentially bumping our stage back
on screen
        stage.regX = stage.regX - (localWalls.bottomRightCorner.x -
stage.canvas.width);//Do the save thing for regX (essentially our
viewport) except use local walls because we set reg to local and stage
to global (above code)
    }
    else {
        //No changes needed
    }

    if (localWalls.topLeftCorner.y < 0) //If overflow on the bottom
(backwards I know)
    {
        stage.y = stage.y + globalWalls.topLeftCorner.y; //Since
topLeftCorner.y is always negative, we're subtracting how far over we
are from the stage, essentially bumping our stage back on screen
        stage.regY = stage.regY - localWalls.topLeftCorner.y;//Do the
save thing for regY (essentially our viewport) except use local walls
because we set reg to local and stage to global (above code)

    }
    else if (localWalls.bottomRightCorner.y > stage.canvas.height) //If
overflow on the top (backwards I know)
    {
        stage.y = stage.y + (globalWalls.bottomRightCorner.y -
stage.canvas.height); //Subtract the height, so we get just the amount
```

```
that we're over, then add to stage, essentially bumping our stage back
on screen
        stage.regY = stage.regY - (localWalls.bottomRightCorner.y -
stage.canvas.height);//Do the save thing for regY (essentially our
viewport) except use local walls because we set reg to local and stage
to global (above code)
    }
    else {
        //No changes needed
    }
}

function drawLoadedDrawings(fileLine)
{
    /***********************Used this website to determine how to grab
what I wanted from each string***********************/
    // http://stackoverflow.com/questions/9133102/how-to-grab-
substring-before-a-specified-character-jquery-or-javascript //

/**********************************************************************
************************************************/
    var shapeID = fileLine.substring(0, fileLine.indexOf(','));
    var restOfShape = fileLine.substring((shapeID.length+1),
(fileLine.indexOf('\r')+1)); //plus 1 will also remove the old comma
left over from grabbing shapeID
    if(shapeID == POINT)
    {
        if (!loadPoint(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else
        {
            return true;
        }
    }
    else if (shapeID == PENCIL)
    {
        if (!loadPencil(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else if (shapeID == LINE)
    {
        if (!loadLine(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else if (shapeID == POLYLINE)
    {
        if (!loadPolyline(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else if (shapeID == POLYGON)
    {
        if (!loadPolygon(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else if (shapeID == CIRCLE)
    {
        if (!loadCircle(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else if (shapeID == AUTO_CLUSTER)
    {
        if (!loadAuto_Cluster(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else if(shapeID == FLOOD_FILL)
    {
        if (!loadFlood_Fill(restOfShape)) //If bad data
        {
            return false; //Stop the load process
        }
        else {
            return true;
        }
    }
    else
    {
        switchAlertLabelForAlert();
        alertify.alert("No shape ID was found, please make sure you are
loading the correct file and then try again.");
        switchAlertLabelForConfirm();
    }
    stage.update();
```

```
}

function loadPoint(shapeInfo)
{
    var label, color, x, y, testYvalue; //Values to be taken from
shapeInfo
    var testInput;

    label = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
label
    shapeInfo = shapeInfo.substring((label.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the label from our string and
the comma after it
    color = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
color
    shapeInfo = shapeInfo.substring((color.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    x = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the x
coordinate
    shapeInfo = shapeInfo.substring((x.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the x coordinate from our
string and the comma after it
    testYvalue = shapeInfo.indexOf(','); //Check to see if there extra
commas (some csv files add them automatically)
    if (testYvalue == -1) //if there aren't excessive commas at the end
of our file
    {
        y = shapeInfo.substring(0, shapeInfo.indexOf('\r')); //Grab the
endY coordinate  (No need to remove this from shapeInfo as its the last
thing we grab)
    }
    else //Our file contains extra commas at the end of this line
    {
        y = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
endY coordinate  (No need to remove this from shapeInfo as its the last
thing we grab)
    }

    //Need to do data cleaning on label, color, x and y
    if (cleanLoadDataLCXY(label, color, x, y))
    {
        /*Start creating our shape now that we have the values we need
and they are valid values*/
        var point = new createjs.Shape();
        stage.addChild(point);
        point.name = Math.floor(Date.now() * Math.random());
        point.id = POINT;
        point.shadow = new createjs.Shadow("#000", 0, 0, 0); //Create a
new shadow that will display if the user selects this shape after
creating it to show the user that this shape is currently selected
        point.graphics.setStrokeStyle(.25); //Line width
        point.graphics.beginStroke(color);
        point.graphics.beginFill(color);
        point.graphics.drawRect(x, y, 1, 1); //Places 1 pixel dot at
location of mouse
```

```
        var newPoint = new pointConstr(null, x, y, color, point);
//Create our point with the data gathered from our file
        drawingData.pointData.push(newPoint); //Push object onto the
point array
        drawingData.undoDrawingOrder.push(POINT); //Pushing a POINT
means the pointTool was used, so if we undo any drawings, we undo from
the pointData array
        addLabelToDropDownMenu(label); //Add label to drop down menu if
it hasn't been added already
        addLabelToDrawings(label); //Add label to our label count which
helps in naming the csv file and maintaining the dropDownMenu labels
        if ($("#labelDropDown").hasClass('show'))
        {
            addLabelToDropDownMenu(label);
        }

/***********************************************************************
*****************/
        return true; //Success
    }
    else
    {
        return false; //failed due to bad data
    }
}
function loadPencil(shapeInfo)
{
    var label, color;
    var x = [];
    var y = [];
    var testYvalue; //Look to see if indexOf returns -1, if so, use
'/r' instead of ','
    label = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
label
    shapeInfo = shapeInfo.substring((label.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the label from our string and
the comma after it
    color = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
color
    shapeInfo = shapeInfo.substring((color.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    if (cleanLoadDataLC(label, color))
    {
        while (shapeInfo) {

            x.push(shapeInfo.substring(0, shapeInfo.indexOf(',')));
//Grab the x coordinate
            shapeInfo = shapeInfo.substring((x[x.length - 1].length +
1), (shapeInfo.indexOf('\r') + 1)); //Remove the x coordinate from our
string and the comma after it
            testYvalue = shapeInfo.indexOf(',');
            if (testYvalue == -1) {
                y.push(shapeInfo.substring(0,
shapeInfo.indexOf('\r'))); //Grab the y coordinate (use \r because this
is the last value in our string
            }
            else {
                y.push(shapeInfo.substring(0, shapeInfo.indexOf(',')));
//Grab the y coordinate
            }
            if (y[y.length - 1] === "") //If we grabed nothing due to a
file containing excess commas
            {
                /*pop blank spaces off our array (should just have 1 on
each) and get us out of the while loop*/
                y.pop();
                x.pop();
                break;

/***********************************************************************
******************/
            }
            shapeInfo = shapeInfo.substring((y[y.length - 1].length +
1), (shapeInfo.indexOf('\r') + 1)); //Remove the y coordinate from our
string and the comma after it
            if(cleanLoadDataXY(x[x.length-1], y[y.length-1])) //If
clean data
            {
                //keep going
            }
            else //quit
            {
                return false; //because bad data
            }

        }

        var pencil = new createjs.Shape();
        var newPencil = new pencilConstr(null, color, pencil, x[0],
y[0]); //Create our pencil drawing with data from our file
        drawingData.undoDrawingOrder.push(PENCIL); //Pushing a LINE
means the lineTool was used, so if we undo any drawings, we undo from
the lineData array
        drawingData.pencilData.push(newPencil);

        pencil.name = Math.floor(Date.now() * Math.random());
        pencil.id = PENCIL;
        pencil.shadow = new createjs.Shadow("#000", 0, 0, 0); //Create
a new shadow that will display if the user selects this shape after
creating it to show the user that this shape is currently selected
        pencil.graphics.setStrokeStyle(1);
        pencil.graphics.beginStroke(color);
        pencil.graphics.moveTo(x[0], y[0]); //0 is our first point,
thus start at 1 for j
        for (var j = 1; j < x.length; j++) {
            pencil.graphics.lineTo(x[j], y[j]);
        }
        var arrayCounter = 0;
        pencil.graphics.endStroke();
        while (x.length > arrayCounter) //Update the drawingData array
with our pencil vertices
        {
            drawingData.pencilData[drawingData.pencilData.length -
1].x[arrayCounter] = x[arrayCounter];
            drawingData.pencilData[drawingData.pencilData.length -
1].y[arrayCounter] = y[arrayCounter];
            arrayCounter++;
        }
        drawingData.pencilData[drawingData.pencilData.length -
1].pencil = pencil;

stage.addChild(drawingData.pencilData[drawingData.pencilData.length -
1].pencil);
        addLabelToDropDownMenu(label);
        addLabelToDrawings(label); //Add label to our label count which
helps in naming the csv file and maintaining the dropDownMenu labels
        if ($("#labelDropDown").hasClass('show')) {
            addLabelToDropDownMenu(label);
        }
        return true; //Data is clean
    }
    else //label or color was not clean data
    {
        return false;
    }
}
function loadLine(shapeInfo)
{
    var label, color, startX, startY, endX, endY, testYvalue;
    label = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
label
    shapeInfo = shapeInfo.substring((label.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the label from our string and
the comma after it
    color = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
color
    shapeInfo = shapeInfo.substring((color.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    startX = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
start x coordinate
    shapeInfo = shapeInfo.substring((startX.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the startX coordinate from our
string and the comma after it
    startY = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
start y coordinate
    shapeInfo = shapeInfo.substring((startY.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the startY coordinate from our
string and the comma after it
    endX = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
endX coordinate
    shapeInfo = shapeInfo.substring((endX.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the endX coordinate from our
string and the comma after it
    testYvalue = shapeInfo.indexOf(',');
    if (testYvalue == -1) //if there aren't excessive commas at the end
of our file
    {
        endY = shapeInfo.substring(0, shapeInfo.indexOf('\r')); //Grab
the endY coordinate  (No need to remove this from shapeInfo as its the
last thing we grab)
    }
    else //Our file contains extra commas at the end of this line
    {
        endY = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab
the endY coordinate  (No need to remove this from shapeInfo as its the
last thing we grab)
    }
    /****Spilt into two functions so i don't have to create a fourth
cleanLoadData to handle 6 inputs****/
    if (cleanLoadDataLCXY(label, color, startX, startY))
    {
        if(cleanLoadDataXY(endX, endY))
        {

/***********************************************************************
******************************/

            /*Start creating our shape now that we have the values we
need and they are valid values*/
            var line = new createjs.Shape();
            stage.addChild(line);
            line.name = Math.floor(Date.now() * Math.random());
            line.id = LINE;
            line.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
            line.graphics.setStrokeStyle(1); //Line width
            line.graphics.beginStroke(color);
            line.graphics.moveTo(startX, startY);
            line.graphics.lineTo(endX, endY);
            line.graphics.endStroke();

            var newLine = new lineConstr(null, color, line, startX,
startY, endX, endY); //label is null because the user has yet to set
it, pass the line object so I can remove it later
            drawingData.undoDrawingOrder.push(LINE); //Pushing a LINE
means the lineTool was used, so if we undo any drawings, we undo from
the lineData array
            drawingData.lineData.push(newLine);
            addLabelToDropDownMenu(label);
            addLabelToDrawings(label); //Add label to our label count
which helps in naming the csv file and maintaining the dropDownMenu
labels
            if ($("#labelDropDown").hasClass('show')) {
                addLabelToDropDownMenu(label);
            }

/***********************************************************************
******************/
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
    return true;
}
function loadPolyline(shapeInfo)
{
    var label, color;
    var x = [];
    var y = [];
    var testYvalue; //Look to see if indexOf returns -1, if so, use
'/r' instead of ','
    label = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
label
    shapeInfo = shapeInfo.substring((label.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the label from our string and
the comma after it
    color = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
color
    shapeInfo = shapeInfo.substring((color.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    if (cleanLoadDataLC(label, color))
    {
        while (shapeInfo)
        {
            x.push(shapeInfo.substring(0, shapeInfo.indexOf(',')));
//Grab the x coordinate
            shapeInfo = shapeInfo.substring((x[x.length - 1].length +
1), (shapeInfo.indexOf('\r') + 1)); //Remove the x coordinate from our
string and the comma after it
            testYvalue = shapeInfo.indexOf(',');
            if (testYvalue == -1) {
                y.push(shapeInfo.substring(0,
shapeInfo.indexOf('\r'))); //Grab the y coordinate (use \r because this
is the last value in our string
            }
            else {
                y.push(shapeInfo.substring(0, shapeInfo.indexOf(',')));
//Grab the y coordinate (use \r because this is the last value in our
string
            }
            if (y[y.length - 1] === "") //If we grabed nothing due to a
file containing excess commas
            {
                /*pop blank spaces off our array (should just have 1 on
each) and get us out of the while loop*/
                y.pop();
                x.pop();
                break;

/***********************************************************************
******************/
            }
            shapeInfo = shapeInfo.substring((y[y.length - 1].length +
1), (shapeInfo.indexOf('\r') + 1)); //Remove the y coordinate from our
string and the comma after it
            if (cleanLoadDataXY(x[x.length-1], y[y.length- 1])) //If
good x, y data
            {
                //Allow to continue
            }
            else //bad data
            {
                return false;
            }
        }
        var polyline = new createjs.Shape();
        polyline.shadow = new createjs.Shadow("#000", 0, 0, 0);
//Create a new shadow that will display if the user selects this shape
after creating it to show the user that this shape is currently
selected
        var newPolyline = new polylineConstr(null, color, polyline,
x[0], y[0], x[1], y[1]); //label is null because the user has yet to
set it, pass the line object so I can remove it later
        drawingData.undoDrawingOrder.push(POLYLINE); //Pushing a
POLYLINE means the polylineTool was used, so if we undo any drawings,
we undo from the polylineData array
        drawingData.polylineData.push(newPolyline);
        var arrayCounter = 2; //2 because we pushed 2 points already
onto the polyline
        while (x.length > arrayCounter) //While not everything has been
transferred over to polylineData
        {
            drawingData.polylineData[drawingData.polylineData.length -
1].x.push(x[arrayCounter]);
            drawingData.polylineData[drawingData.polylineData.length -
1].y.push(y[arrayCounter]);
            arrayCounter++;
        }
        addLabelToDropDownMenu(label);
        addLabelToDrawings(label); //Add label to our label count which
helps in naming the csv file and maintaining the dropDownMenu labels
        drawCompletePolyline(true, -1); //Draws the completed polyline
for us -> -1 so we know how to treat these lines when drawing
        if ($("#labelDropDown").hasClass('show')) {
            addLabelToDropDownMenu(label);
        }
    }
    else
    {
        return false; //bad data
    }
    return true; //good data
}
function loadPolygon(shapeInfo)
{
    var label, color, fillColor;
    var x = [];
    var y = [];
    var testYvalue; //Look to see if indexOf returns -1, if so, use
'/r' instead of ','
    label = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
label
    shapeInfo = shapeInfo.substring((label.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the label from our string and
the comma after it
    color = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
color
    shapeInfo = shapeInfo.substring((color.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    fillColor = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab
the color
    shapeInfo = shapeInfo.substring((fillColor.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    if (cleanLoadDataLCF(label, color, fillColor)) //Good data so far
    {
        while (shapeInfo) {
            x.push(shapeInfo.substring(0, shapeInfo.indexOf(',')));
//Grab the x coordinate
            shapeInfo = shapeInfo.substring((x[x.length - 1].length +
1), (shapeInfo.indexOf('\r') + 1)); //Remove the x coordinate from our
string and the comma after it
            testYvalue = shapeInfo.indexOf(',');
            if (testYvalue == -1) {
                y.push(shapeInfo.substring(0,
shapeInfo.indexOf('\r'))); //Grab the y coordinate (use \r because this
is the last value in our string
            }
            else {
                y.push(shapeInfo.substring(0, shapeInfo.indexOf(',')));
//Grab the y coordinate (use \r because this is the last value in our
string
            }
            if (y[y.length - 1] === "") //If we grabed nothing due to a
file containing excess commas
            {
                /*pop blank spaces off our array (should just have 1 on
each) and get us out of the while loop*/
                y.pop();
                x.pop();
                break;

/***********************************************************************
******************/
            }
            shapeInfo = shapeInfo.substring((y[y.length - 1].length +
1), (shapeInfo.indexOf('\r') + 1)); //Remove the y coordinate from our
string and the comma after it
            if (cleanLoadDataXY(x[x.length-1], y[y.length-1])) //If
good data
            {
                //Keep going
            }
            else //Bad data
            {
                return false;
            }
        }
        var polygon = new createjs.Shape();
        polygon.shadow = new createjs.Shadow("#000", 0, 0, 0); //Create
a new shadow that will display if the user selects this shape after
creating it to show the user that this shape is currently selected
        var newPolygon = new polygonConstr(null, color, fillColor,
polygon, x[0], y[0], x[1], y[1]); //label is null because the user has
yet to set it, pass the line object so I can remove it later
        drawingData.undoDrawingOrder.push(POLYGON); //Pushing a
POLYLINE means the polylineTool was used, so if we undo any drawings,
we undo from the polylineData array
        drawingData.polygonData.push(newPolygon);
        var arrayCounter = 2; //2 because we pushed 2 points already
onto the polyline
        while (x.length > arrayCounter) //While not everything has been
transferred over to polylineData
        {
            drawingData.polygonData[drawingData.polygonData.length -
1].x.push(x[arrayCounter]);
            drawingData.polygonData[drawingData.polygonData.length -
1].y.push(y[arrayCounter]);
            arrayCounter++;
        }
        addLabelToDropDownMenu(label);
        addLabelToDrawings(label); //Add label to our label count which
helps in naming the csv file and maintaining the dropDownMenu labels
        drawCompletePolygon(true, -1); //Draws the completed polygon
for us -> -1 so we know how to treat these lines when drawing
        if ($("#labelDropDown").hasClass('show')) {
            addLabelToDropDownMenu(label);
        }
    }
    else // bad data
    {
        return false;
    }
    return true;
}
function loadCircle(shapeInfo)
{
    var label, color, fillColor, x, y, radius, testRadiusValue;
    label = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
label
    shapeInfo = shapeInfo.substring((label.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the label from our string and
the comma after it
    color = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the
color
    shapeInfo = shapeInfo.substring((color.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the color from our string and
the comma after it
    fillColor = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab
the fillColor
```

```
    shapeInfo = shapeInfo.substring((fillColor.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the fillColor from our string
and the comma after it
    x = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the x
coordinate
    shapeInfo = shapeInfo.substring((x.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the x coordinate from our
string and the comma after it
    y = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab the y
coordinate
    shapeInfo = shapeInfo.substring((y.length + 1),
(shapeInfo.indexOf('\r') + 1)); //Remove the y coordinate from our
string and the comma after it
    testRadiusValue = shapeInfo.indexOf(',');
    if (testRadiusValue == -1) //if there aren't excessive commas at
the end of our file
    {
        radius = shapeInfo.substring(0, shapeInfo.indexOf('\r'));
//Grab the radius (No need to remove this from shapeInfo as its the
last thing we grab)
    }
    else //Our file contains extra commas at the end of this line
    {
        radius = shapeInfo.substring(0, shapeInfo.indexOf(',')); //Grab
the radius (No need to remove this from shapeInfo as its the last thing
we grab)
    }
    if (cleanLoadDataCircle(label, color, fillColor, x, y, radius))
//Validate inputs
    {
        /*Start creating our shape now that we have the values we need
and they are valid values*/
        var circle = new createjs.Shape();
        stage.addChild(circle);
        circle.name = Math.floor(Date.now() * Math.random());
        circle.id = CIRCLE;
        circle.shadow = new createjs.Shadow("#000", 0, 0, 0); //Create
a new shadow that will display if the user selects this shape after
creating it to show the user that this shape is currently selected
        circle.graphics.setStrokeStyle(1); //Line width
        circle.graphics.beginStroke(color);
        circle.graphics.beginFill(hex2rgba(color, fillColor));
        circle.graphics.drawCircle(x, y, radius);
        circle.graphics.endStroke();

        var newCircle = new circleConstr(null, color, fillColor,
circle, x, y, radius); //label is null because the user has yet to set
it, pass the circle object so I can remove it later
        drawingData.undoDrawingOrder.push(CIRCLE); //Pushing a LINE
means the lineTool was used, so if we undo any drawings, we undo from
the lineData array
        drawingData.circleData.push(newCircle);
        addLabelToDropDownMenu(label);
        addLabelToDrawings(label); //Add label to our label count which
helps in naming the csv file and maintaining the dropDownMenu labels
        if ($("#labelDropDown").hasClass('show')) {
            addLabelToDropDownMenu(label);
        }
/***********************************************************************
******************/
    }
    else //bad data
    {
        return false;
    }
    return true; //Good data
}
function loadAuto_Cluster(shapeInfo)
{
    //Can't draw yet, so can't load in
}
function loadFlood_Fill(shapeInfo)
{
    //Can't draw yet, so can't load in
}

function wipeDrawingsForLoad()
{
    stage.removeAllChildren(); //Clears everything on the stage to make
room for the new stage
    stage.addChild(bitmap); //Add our image back on
    myGraphics.removeAllChildren; //remove all shape objects
    myGraphics.clear() //Wipe away everything so we can redraw it at
the correct scale.
    drawingData.wipeAll(); //Empty out all the arrays that help our
drawing information
    turnOffButton();
    stage.update();
    /*Wipe our old labels away from the drop down menu*/
    var dropDownMenu = document.getElementById("dropDown");
    for (var i = dropDownMenu.length; i > 0; i--) //Cycle through the
dropdown menu backwards, ignoring the first one ("All Labels")
    {
        dropDownMenu.remove(i); //Remove the label from the drop down
menu
    }
}

function switchAlertLabelForAlert()
{
    /*Change button to Okay for this alert*/
    alertify.set ({ labels: { ok: "Okay", cancel: "No" } });
    /************************************************/
}
function switchAlertLabelForConfirm()
{
    /*Change button to back to Yes for this confirms*/
    alertify.set ({ labels: { ok: "Yes", cancel: "No" }});
    /************************************************/
}

/***************************Clean Data
Functions****************************/
```

```
function cleanLoadDataLCXY(label, color, x, y)
{
    var testLabel, testColor, testX, testY;
    testLabel = label.replace(/[^a-z'_0-9 ]/ig, ""); //Allow Letters A
- z, ', _, numbers 0 - 9, and nothing else
    if (testLabel != label || testLabel.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
label of a shape is not a legitimate value. Labels can only contain
letters, numbers, underscores, and apostrophes. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testColor = color.replace(/[^#a-f0-9]/ig, ""); //Allow the color to
be in the format of "#000000" only (aka only allow a hex representation
of rgb)
    if (testColor.length != 7) //#xxxxxx
    {
        if(testColor.length !=4) //#xxx
        {
            color = "z"; //z because if there is a z in color it will
get stripped out in color.replace, thus testColor cannot equal color
        }
    }
    if (testColor != color) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
color value of a shape is not a legitimate value. The correct format is
\"#xxxxxx\". Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testX = x.replace(/[^0-9.]/ig, ""); //Allow the code to be any
floating point or integer number, but nothing else
    if (testX != x || testX.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The X
coordinate of a shape is not a legitimate value. Acceptable values
include all real numbers. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testY = y.replace(/[^0-9.]/ig, ""); //Allow the code to be any
floating point or integer number, but nothing else
    if (testY != y || testY.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The Y
coordinate of a shape is not a legitimate value. Acceptable values
include all real numbers. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    return true; //Data is clean
}
function cleanLoadDataLC(label, color)
{
    var testLabel, testColor;
    testLabel = label.replace(/[^a-z'_0-9 ]/ig, ""); //Allow Letters A
- z, ', _, numbers 0 - 9, and nothing else
    if (testLabel != label || testLabel.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
label of a shape is not a legitimate value. Labels can only contain
letters, numbers, underscores, and apostrophes. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testColor = color.replace(/[^#a-f0-9]/ig, ""); //Allow the color to
be in the format of "#000000" only (aka only allow a hex representation
of rgb)
    if (testColor.length != 7) //#xxxxxx
    {
        if (testColor.length != 4) //#xxx
        {
            color = "z"; //z because if there is a z in color it will
get stripped out in color.replace, thus testColor cannot equal color
        }
    }
    if (testColor != color) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
color value of a shape is not a legitimate value. The correct format is
\"#xxxxxx\". Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    return true; //Data is clean
}
function cleanLoadDataXY(x, y)
{
    var testX, testY;
    testX = x.replace(/[^0-9.]/ig, ""); //Allow the code to be any
floating point or integer number, but nothing else
    if (testX != x || testX.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The X
coordinate of a shape is not a legitimate value. Acceptable values
include all real numbers. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testY = y.replace(/[^0-9.]/ig, ""); //Allow the code to be any
floating point or integer number, but nothing else
    if (testY != y || testY.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The Y
coordinate of a shape is not a legitimate value. Acceptable values
include all real numbers. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    return true; //Data is clean
}
function cleanLoadDataLCF(label, color, fillColor) //Currently Polygon
specific
{
    var testLabel, testColor, testFillColor;
    testLabel = label.replace(/[^a-z'_0-9 ]/ig, ""); //Allow Letters A
- z, ', _, numbers 0 - 9, and nothing else
    if (testLabel != label || testLabel.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
label of a polygon is not a legitimate value. Labels can only contain
letters, numbers, underscores, and apostrophes. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testColor = color.replace(/[^#a-f0-9]/ig, ""); //Allow the color to
be in the format of "#000000" only (aka only allow a hex representation
of rgb)
    if (testColor.length != 7) //#xxxxxx
    {
        if (testColor.length != 4) //#xxx
        {
            color = "z"; //z because if there is a z in color it will
get stripped out in color.replace, thus testColor cannot equal color
        }
    }
    if (testColor != color) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
color value of a polygon is not a legitimate value. The correct format
is \"#xxxxxx\", where x is a hexidecimal. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testFillColor = fillColor.replace(/[^0-9.]/ig, ""); //Allow the
fill color (Which is just the alpha channel) to be in the format of
"x.xx"
    if (testFillColor.length <= 2) //.x
    {
        fillColor = "z"; //z because fillColor can't be z, thus making
testFillColor != fillColor true
    }
    if (testFillColor > 1 || testFillColor < 0) //if not a value
between 0 and 1
    {
        fillColor = "z";//z because fillColor can't be z, thus making
testFillColor != fillColor true
    }
    if (testFillColor != fillColor || testFillColor.length > 8) //Bad
data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
alpha channel of your fill color for a polygon is not a legitimate
value. Acceptable values are any number between 0 and 1 and less than 8
digits. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    return true; //Data is clean
}
function cleanLoadDataCircle(label, color, fillColor, x, y, radius)
//Currently circle specific
{
    var testLabel, testColor, testFillColor, testX, testY, testRadius;
    testLabel = label.replace(/[^a-z'_0-9 ]/ig, ""); //Allow Letters A
- z, ', _, numbers 0 - 9, and nothing else
    if (testLabel != label || testLabel.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
label of a circle is not a legitimate label. Labels can only contain
letters, numbers, underscores, and apostrophes. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testColor = color.replace(/[^#a-f0-9]/ig, ""); //Allow the color to
be in the format of "#000000" only (aka only allow a hex representation
of rgb)
    if (testColor.length != 7) //#xxxxxx
    {
        if (testColor.length != 4) //#xxx
        {
            color = "z"; //z because if there is a z in color it will
get stripped out in color.replace, thus testColor cannot equal color
        }
    }
    if (testColor != color) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
color value of a circle is not a legitimate value. The correct format
is \"#xxxxxx\". Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testFillColor = fillColor.replace(/[^0-9.]/ig, ""); //Allow the
fill color (Which is just the alpha channel) to be in the format of
"x.xx"
    if (testFillColor.length <= 2) //.x
    {
        fillColor = "z"; //z because fillColor can't be z, thus making
testFillColor != fillColor true
    }
    if (testFillColor > 1 || testFillColor < 0) //if not a value
between 0 and 1
    {
        fillColor = "z";//z because fillColor can't be z, thus making
testFillColor != fillColor true
    }
    if (testFillColor != fillColor || testFillColor.length > 8) //Bad
data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
alpha channel of your fill color for a circle is not a legitimate
value. Acceptable values are any number between 0 and 1 and less than 8
digits. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testX = x.replace(/[^0-9.]/ig, ""); //Allow the code to be any
floating point or integer number, but nothing else
    if (testX != x || testX.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The X
coordinate of a circle is not a legitimate value. Acceptable values
include all real numbers. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testY = y.replace(/[^0-9.]/ig, ""); //Allow the code to be any
floating point or integer number, but nothing else
    if (testY != y || testY.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The Y
coordinate of a circle is not a legitimate value. Acceptable values
include all real numbers. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    testRadius = radius.replace(/[^0-9.]/ig, ""); //Allow the fill
color (Which is just the alpha channel) to be in the format of "x.xx"
    if (testRadius != radius || testRadius.length == 0) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
radius of this circle is not a legitimate value. Cancelling load...");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }
    return true; //Data is clean
}
/************************************************************************
*****/
function addTooltip(arrayPosition, drawingType) //Place the label of
the image where the user clicked on the shape
{
    if (drawingType[arrayPosition].label == null)
    {
        labelToolTip.text = "This shape has yet to be labeled.";
    }
    else
    {
        labelToolTip.text = drawingType[arrayPosition].label;
    }
    var mouseCoordinates = stage.globalToLocal(stage.mouseX,
stage.mouseY);
    labelToolTip.x = mouseCoordinates.x;
    labelToolTip.y = mouseCoordinates.y;
    stage.addChild(labelToolTip);
}
function removeTooltip()
{
    //Removes tooltip
    stage.removeChild(labelToolTip);
}
function unSelectedObject()
{
    if (drawingData.selectedObject[0] != 0) //If we are currently
selecting something -> Stop selecting it so we can select something
else (More like, stop showing we are selecting it)
    {
        if (drawingData.selectedObject[0] == POINT) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            if (drawingData.selectedObject[1] <
drawingData.pointData.length) //If we didn't just delete a shape
            {

drawingData.pointData[drawingData.selectedObject[1]].point.shadow.setTr
ansparent(); // Removes shadow (only need to do this if user clicks
anywhere but the delete button)
                removeTooltip(); //Remove tooltip
            }
        }
        if (drawingData.selectedObject[0] == PENCIL) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            if (drawingData.selectedObject[1] <
drawingData.pencilData.length) //If we didn't just delete a shape
            {

drawingData.pencilData[drawingData.selectedObject[1]].pencil.shadow.set
Transparent(); // Removes shadow (only need to do this if user clicks
anywhere but the delete button)
                removeTooltip();//Remove tooltip
            }
        }
        if (drawingData.selectedObject[0] == LINE) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            if (drawingData.selectedObject[1] <
drawingData.lineData.length) //If we didn't just delete a shape
            {

drawingData.lineData[drawingData.selectedObject[1]].lineObject.shadow.s
etTransparent(); // Removes shadow (only need to do this if user clicks
anywhere but the delete button)
                removeTooltip();//Remove tooltip
            }
        }
        if (drawingData.selectedObject[0] == POLYLINE) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            if (drawingData.selectedObject[1] <
drawingData.polylineData.length) //If we didn't just delete a shape
            {

drawingData.polylineData[drawingData.selectedObject[1]].polyline.shadow
.setTransparent(); // Removes shadow (only need to do this if user
clicks anywhere but the delete button)
                removeTooltip();//Remove tooltip
            }
        }
        if (drawingData.selectedObject[0] == POLYGON) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            if (drawingData.selectedObject[1] <
drawingData.polygonData.length) //If we didn't just delete a shape
            {

drawingData.polygonData[drawingData.selectedObject[1]].polygon.shadow.s
etTransparent(); // Removes shadow (only need to do this if user clicks
anywhere but the delete button)
                removeTooltip();//Remove tooltip
            }
        }
        if (drawingData.selectedObject[0] == CIRCLE) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            if (drawingData.selectedObject[1] <
drawingData.circleData.length) //If we didn't just delete a shape
            {

drawingData.circleData[drawingData.selectedObject[1]].circle.shadow.set
Transparent(); // Removes shadow (only need to do this if user clicks
anywhere but the delete button)
                removeTooltip();//Remove tooltip
            }
        }
        if (drawingData.selectedObject[0] == AUTO_CLUSTER) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            alert("This feature is not yet implemented yet");
        }
        if (drawingData.selectedObject[0] == FLOOD_FILL) //If we are
currently selecting a point -> Stop showing the user we're selecting
that point
        {
            alert("This feature is not yet implemented yet");
        }
        stage.update();
    }
}

function displayShapeByLabel()
{
    var dropDownMenu = document.getElementById("dropDown");
    var selectedValues =
dropDownMenu.options[dropDownMenu.selectedIndex].value; //returns the
option names in the dropdown menu
    stage.removeAllChildren();
    stage.addChild(bitmap);
    for (var i = 0; i < drawingData.pointData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.pointData[i].label == selectedValues)
        {
            stage.addChild(drawingData.pointData[i].point);
        }
    }
    for (var i = 0; i < drawingData.pencilData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.pencilData[i].label == selectedValues) {
            stage.addChild(drawingData.pencilData[i].pencil);
        }
    }
    for (var i = 0; i < drawingData.lineData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.lineData[i].label == selectedValues) {
            stage.addChild(drawingData.lineData[i].lineObject);
        }
    }
    for (var i = 0; i < drawingData.polylineData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.polylineData[i].label == selectedValues) {
            stage.addChild(drawingData.polylineData[i].polyline);
        }
    }
    for (var i = 0; i < drawingData.polygonData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.polygonData[i].label == selectedValues) {
            stage.addChild(drawingData.polygonData[i].polygon);
        }
    }
    for (var i = 0; i < drawingData.circleData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.circleData[i].label == selectedValues) {
            stage.addChild(drawingData.circleData[i].circle);
        }
    }
    for (var i = 0; i < drawingData.auto_clusterData.length; i++)
//Cycle through and only add drawings that have the same label as the
selected label
    {
        if (drawingData.auto_clusterData[i].label == selectedValues) {

stage.addChild(drawingData.auto_clusterData[i].auto_cluster);
        }
    }
    for (var i = 0; i < drawingData.flood_fillData.length; i++) //Cycle
through and only add drawings that have the same label as the selected
label
    {
        if (drawingData.flood_fillData[i].label == selectedValues) {
            stage.addChild(drawingData.flood_fillData[i].flood_fill);
        }
    }
    if (selectedValues == "All Labels")
    {
        resize();
    }
    //Don't want to be selecting an image that is no longer being
displayed so we remove tooltip and unselect shape
    removeTooltip();
    unSelectedObject();
    stage.update();
}
function currentLabelBeingShown()
{
    var dropDownMenu = document.getElementById("dropDown");
    var selectedValues =
dropDownMenu.options[dropDownMenu.selectedIndex].value; //returns the
option names in the dropdown menu
    if (selectedValues == "All Labels")
    {
        return true; //Can do whatever we want
    }
    else
    {
        alertify.error("You must be showing 'All Labels' on the
'Showing:' drop down menu in order to do this.");
        return false; //Can't do things until user reselects "All
Labels".
    }
}

/********Color Wheel Functions*********/
function assignColor(e) //Assign color wheel color to a user button
{
    e.preventDefault(); //Prevent page reload
    //This function (cleaning color wheel input) could be called twice,
(if user hits enter instead of clicking button) but preventing this
from occuring by seeing if the data
    //had been cleaned already will add more complexity to the code
than the benefits of performance make worth while. Plus, it shouldn't
take that long to clean.
    if (testColorWheelInput()) //if data is clean, allow assignment to
happen
    {
        var colorValue = document.getElementById("color").value;
        if (colorValue.length != 7) //if length does not match #xxxxxx
        {
            document.getElementById("color").value =
rgb2hex(document.getElementById("color").style.backgroundColor); //grab
background of textbox as that will still be accurate
            colorValue = document.getElementById("color").value;
//place background of textbox into our colorvalue to still perform
operation later
        }
        var userButtonToChange = "#btn-user" + color.currentUserColor;
//Cycles through the user buttons and sets them, going back to the
first after all have been set
        color.currentUserColor++;
        if (color.currentUserColor > 4) {
            color.currentUserColor = 1;
        }
        $(userButtonToChange).css('background-color', colorValue);
        $(userButtonToChange).css('color', hex2rgba(colorValue,
.35));//everywhere else .35 is fillColor (and currently vice versa is
true as well)
    }
    else
    {
        alertify.error("Color assignment failed.");
    }
}
function colorWheelText(e)
```

```
{
    e.preventDefault(); //prevent page reload
    if (testColorWheelInput()) //if data is clean, allow assignment to
happen
    {
        document.getElementById("colorPickerBtn").click(); // call
assignColor function
    }
    else
    {
        alertify.error("Color assignment failed.");
    }
}
function testColorWheelInput()
{
    var testColor =
document.getElementById("color").value.replace(/[^#a-f0-9]/ig, "");
//Allow the color to be in the format of "#000000" only (aka only allow
a hex representation of rgb)
    if (testColor != document.getElementById("color").value) //Bad data
    {
        switchAlertLabelForAlert();
        alertify.alert("This textbox contains bad data. The color value
is not a legitimate value. The correct format is \"#xxxxxx\", where x
is a hexidecimal digit. Cancelling color assignment...");
        switchAlertLabelForConfirm();
        return false;
    }
    return true;
}
/**************************************/

/*********Load Attribute Table*********/
function loadAttributeTableBtn() {
    alertify.confirm("Warning: Uploading an attribute table will delete
all previously drawn shapes.", function (e) {
        if (e) {

document.getElementById('attributeTable').addEventListener('change',
loadAttributeTable);
            if (window.File && window.FileReader && window.FileList &&
window.Blob)
            {
                $("#attributeTable").trigger('click');
            }
            else {
                switchAlertLabelForAlert();
                alertify.alert("We're sorry, but the File APIs are not
fully supported in this browser Please try a different browser.");
                switchAlertLabelForConfirm();
            }
        }
        else {
            //Do nothing -> its as if they never hit the Select
Attribute Table button in the first place
        }
    });
```

```
}
function loadAttributeTable(e)
{
    /*************************Code for reading a csv file was modified
from these two websites**************************/
    //http://www.htmlgoodies.com/beyond/javascript/read-text-files-
using-the-javascript-filereader.html#fbid=vuei5iprIXD /
    ///http://stackoverflow.com/questions/23331546/how-to-use-
javascript-to-read-local-text-file-and-read-line-by-line ///

/**********************************************************************
**********************************************/
    var drawingDataFile = e.target.files[0];     //Retrieve the first
(and only!) File from the FileList object

    if (!drawingDataFile) {
        switchAlertLabelForAlert();
        alertify.alert("Failed to load file");
        switchAlertLabelForConfirm();
    }
    else if (drawingDataFile.type.match('text/csv.*') ||
drawingDataFile.type.match('text/comma-seperated-values.*') ||
drawingDataFile.type.match('application/vnd.ms-excel.*')
            || drawingDataFile.type.match('application/csv.*') ||
drawingDataFile.type.match('application/excel.*') ||
drawingDataFile.type.match('application/vnd.msexcel.*')) //If the file
is a csv file
    {
        //Resize image to original image size if drawn already
        //Just resizing the image to the original size on load is
faster than resizing every single drawing in our load file to whatever
size the user currently has the image at
        if (originalHeight != 0)
        {
            //I delete all current drawings so we don't have to worry
about bad labels
            wipeDrawingsForLoad(); //remove all currently drawn data so
we don't have to worry about overlaping drawings (or user loading in
more than 1 drawing data set)
            if (potraitLayout) {
                document.getElementById("size").value = originalHeight;
//Set slider bar to correct size
            }
            else {
                document.getElementById("size").value = originalWidth;
//Set slider bar to correct size
            }
            if (stage.canvas.width != originalWidth) //If at a
different size than original, then resample local x,y to match original
            {
                stage.canvas.width = originalWidth; //set canvas to
correct size
            }
            if (stage.canvas.height != originalHeight) {
                stage.canvas.height = originalHeight; //set canvas to
correct size
            }
            resize(); //Now that canvas size is reset, reset image size
        }
        document.getElementById("spinner").style.visibility =
"visible";
        //Begin file read
        var tableInfo = new FileReader();
        tableInfo.onload = function (event) {
            var contents = this.result.split('\n');
            for (var line = 0; line < contents.length; line++) // start
at 1 because we don't care how many shapes there are (first entry in
file is number of shapes)
            {
                loadAttributeTableData(contents[line]);
            }
            document.getElementById("spinner").style.visibility =
"hidden"; //Hide the spinner to let user know we're done loading
        };
        tableInfo.readAsText(drawingDataFile);
        //Make correct labeling system visible to user
        $("#labelOptionsTxt").addClass("show");
        $("#labelOptionsTxt").removeClass("hide");
        $("#labelDropDown").addClass("show");
        $("#labelDropDown").removeClass("hide");
        $("#label").addClass("hide");
        $("#label").removeClass("show");
    }
    else //Not a csv file
    {
        switchAlertLabelForAlert();
        alertify.alert(drawingDataFile.name + " is not a valid csv
file.");
        switchAlertLabelForConfirm();
    }

}
function loadAttributeTableData(fileLine)
{
    /**********************Used this website to determine how to grab
what I wanted from each string**********************/
    // http://stackoverflow.com/questions/9133102/how-to-grab-
substring-before-a-specified-character-jquery-or-javascript //

/***********************************************************************
************************************************/
    var labelID = fileLine.substring(0, fileLine.indexOf(','));
    var labelName = fileLine.substring(labelID.length + 1); //plus 1
will also remove the old comma left over from grabbing shapeID
    if (labelName.substring(0, labelName.indexOf(',')) != "")
    {
        labelName = labelName.substring(0, labelName.indexOf(','));
    }
    var testLabel = labelName.replace(/[^a-z'_0-9 ]/ig, ""); //Allow
Letters A - z, ', _, numbers 0 - 9, and nothing else
    if (testLabel != labelName || testLabel.length == 0) //Bad data
    {
        if (labelName.substring(0, labelName.length -1) == testLabel)
//Needed because sometimes the /n or /r character will still be
attached and trigger a bad label, when its not
        {
            //Label is actually fine, continue as normal and then
return out
            addLabelToLabelDropDown(labelName);
            return;
        }
        switchAlertLabelForAlert();
        alertify.alert("The file you loaded in contains bad data. The
label of a shape is not a legitimate value. Labels can only contain
letters, numbers, underscores, and apostrophes. This label will not be
included");
        alertify.error("Load failed.");
        switchAlertLabelForConfirm();
        return false;
    }

    addLabelToLabelDropDown(labelName);
}
function addLabelToLabelDropDown(labelName)
{
    var labelDropDownMenu = document.getElementById("labelDropDown");
    var option = document.createElement("option");
    option.text = labelName.value || labelName;
    labelDropDownMenu.add(option);
}
/***************************************/
```

create_account.js

```
'use strict';


function validateForm()
{
    if (document.getElementById("userPassword").value == "" ||
document.getElementById("userEmail").value == "") {
        alertify.error("You must fill in all of the fields before
submitting...");
    }
    else {
        //Clean data before sending
        var emailTest =
document.getElementById("userEmail").value.replace(/[^a-z,.@!#*'_0-9
]/ig, "");
        var passwordTest =
document.getElementById("userPassword").value.replace(/[^a-z,.@!#*'_0-9
]/ig, "");
        var badData = false;
        if (emailTest != document.getElementById("userEmail").value ||
emailTest.length == 0) {
            badData = true;
        }
        if (passwordTest !=
document.getElementById("userPassword").value || passwordTest.length ==
0) {
            badData = true;
        }
        if (badData) {
            alertify.error("Email or password contained bad data.
Please try again.");
        }
        else {
            document.getElementById("LoginSubmit").submit();
        }
    }
}

function checkName(nameParameter) {
    var name = document.getElementById(nameParameter);
    var nameValue = name.value;
    var re = /^[0-9A-Za-z ]+$/;
    if (re.test(nameValue)) {
        name.className = 'LV_valid_field';
    }
    else {
        name.className = 'LV_invalid_field';
    }
}

function checkEmail() {
    var userEmail = document.getElementById('userEmail');
    var emailValue = userEmail.value;
    var re = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-
z]{2,6}(?:\.[a-z]{2})?)$/i;
    if (re.test(emailValue)) {
        userEmail.className = 'LV_valid_field';
    }
    else {
        userEmail.className = 'LV_invalid_field';
    }
};

function checkFilled(elementID) {
    var wantedDiv = document.getElementById(elementID);
    var wantedValue = wantedDiv.value;
    if (wantedValue.length !== 0) {
        wantedDiv.className = 'LV_valid_field';
    }
    else {
        wantedDiv.className = 'LV_invalid_field';
    }
}

function checkPass(prefixPara) {
    //Store the password field objects into variables ...
    var passOriginal = document.getElementById(prefixPara +
'Password');
    var passAuthenticate = document.getElementById(prefixPara +
'PasswordAuth');
```

```
    //check if password authenticate is same as original, if so set
class to 'LV_valid_field' else make it invalid
    if (passOriginal.value === passAuthenticate.value &&
passOriginal.value.length !== 0) {
        passAuthenticate.className = 'LV_valid_field';
    }
    else {
        passAuthenticate.className = 'LV_invalid_field';
    }
};

function submitValidationHandler() {
    var firstName = document.getElementById('firstName');
    var lastName = document.getElementById('lastName');
    var userEmail = document.getElementById('userEmail');
    var userPassword = document.getElementById('userPassword');
    var userPassAuth = document.getElementById('userPasswordAuth');
    var orgName = document.getElementById('orgName');
    var federalYes = document.getElementById('federalYes').checked;
    var federalNo = document.getElementById('federalNo').checked;
    var errorFound = false;
    if (firstName.className !== 'LV_valid_field')
    {
        errorFound = true;
        alertify.error('Name cannot contain special characters or be
blank.');
    }
    if (lastName.className !== 'LV_valid_field')
    {
        errorFound = true;
        alertify.error('Name cannot contain special characters or be
blank.');
    }
    if (userEmail.className !== 'LV_valid_field')
    {
        errorFound = true;
        alertify.error('Not a valid email address.');
    }
    if (userPassword.className !== 'LV_valid_field')
    {
        errorFound = true;
        alertify.error('Password field cannot be blank.');
    }
    if (userPassAuth.className !== 'LV_valid_field')
    {
        errorFound = true;
        alertify.error('Passwords do not match.');
    }
    if (orgName.className !== 'LV_valid_field')
    {
        errorFound = true;
        alertify.error('Organization name cannot contain special
characters or be blank.');
    }
    if (!federalYes && !federalNo)
    {
        errorFound = true;
        alertify.error('Federal field needs to be filled out.');
    }
    if (errorFound === false)
    {
        document.getElementById("CreateAccount").submit();
    }

};
```

layout.js

```
'use strict';
function logoutCall()
{
    document.getElementById("logoutForm").submit();
}
```

views

create_account_page.vash

```
@html.extend('layout', function(model)
{
    @html.block("cssFiles", function(model)
    {
        <link href="/css/create_account.css" rel="stylesheet"/>
    })
    @html.block("body", function(model)
    {

        @if (model.message)
        {
            <p>@model.message</p>
        }

        <div class="row">
            <div class = "centerAlign">
                <form  id="CreateAccount" class = "centerAlign"
action="/create_account" method="post" target="_self">
                    <br>
                    <h3>
                        Contact Information
                    </h3>
                    <p><font color = "#be0f34" size =
"4">*</font><input class = "textBox" type="text" id="firstName"
name="firstName" placeholder = "First Name" default="default"
onkeyup="checkName('firstName'); return false;"
onchange="checkName('firstName');"></p>
                    <p><font color = "#be0f34" size =
"4">*</font><input class = "textBox" type="text" id="lastName"
name="lastName" placeholder = "Last Name" default="default"
onkeyup="checkName('lastName'); return false;"
onchange="checkName('lastName');"></p>
                    <p><font color = "#be0f34" size =
"4">*</font><input class = "textBox" type="text" id="userEmail"
name="userEmail" placeholder = "Email" default="default"
onkeyup="checkEmail(); return false;" onchange="checkEmail();">
                        <a href = "#" class = "toolTip">
                            <img src="/images/tool_tip_icon.jpg"
alt="Tool Tip"/>
                            <span>
                                <img class = "calloutLong" src =
"/images/callout.gif" alt = "Tool Tip"/>
                                <strong>What do we use your email
for?</strong><br/>
                                Your email is how you will log into our
website to upload and download imagery as well as use the Training Data
Selector and view reports.
                            </span>
                        </a>
                    </p>
                    <p><font color = "#be0f34" size =
"4">*</font><input class = "textBox" type="password" id="userPassword"
```

```
name="userPassword" placeholder = "Password" default="default"
onkeyup="checkFilled('userPassword'); return false;"
onchange="checkPass('user');"></p>
                    <p><font color = "#be0f34" size =
"4">*</font><input class = "textBox" type="password"
id="userPasswordAuth" placeholder = "Re-enter Password"
default="default" onkeyup="checkPass('user'); return false;"
onchange="checkPass('user');"></p>
                    <p><font color = "#be0f34" size =
"4">*</font><input class = "textBox" type="text" id="orgName"
name="orgName" placeholder = "Organization Name" default="default"
onkeyup="checkName('orgName'); return false;"
onchange="checkName('orgName');">
                        <a href = "#" class = "toolTip">
                            <img src="/images/tool_tip_icon.jpg"
alt="Tool Tip"/>
                            <span>
                                <img class = "calloutLong" src =
"/images/callout.gif" alt = "Tool Tip"/>
                                <strong>What is this?</strong><br/>
                                Enter the name of the business or
organization you are affiliated with. If you are not affiliated with
any business or organization, then enter "Personal".
                            </span>
                        </a>
                    </p>

                    <font color = "#be0f34" size = "4">*</font>
Federal:
                    <input type="radio" name="federal" id="federalYes"
value="1"> Yes
                    <input type="radio" name="federal" id="federalNo"
value="0"> No

                    <span id = "buttonValidationMessage"> </span>
                    <a href = "#" class = "toolTip federal">
                        <img src="/images/tool_tip_icon.jpg" alt="Tool
Tip"/>
                        <span>
                            <font size = "2">
                                <img class = "calloutMiddle" src
="/images/callout.gif" alt="Tool Tip"/>
                                <strong>What does this mean?</strong>
                                <div class = "removebold">
                                    If you are a Federal Organization
we need to know because it changes how we handle your data.
                                </div>
                            </font>
                        </span>
                    </a>
                    <p><font color = "#be0f34" size = "2">* Required
Fields</font></p>
                    <button type="button" onclick =
'submitValidationHandler()'>Submit</button>
                    <br>
                    <br>
                    <br>
                    <br>
                    <br>
                    <br>
                    <br>
                    <br>
                </form>
            </div>
        </div>
    })


    @html.block("jsFiles", function(model)
    {
        <script type="text/javascript"
src="/lib/alertify/alertify.min.js"></script>
        <script type="text/javascript"
src="/js/create_account.js"></script>
    })

})
```

download_imagery.vash

```
@html.extend('layout', function(model)
{
    @html.block("body", function(model)
    {
        <div class=" col-xs-offset-3 col-xs-6 col-sm-6 col-md-6"
style="text-align: center;">
            <p><br>This page is still being built. Thank you for your
patience.</p>
        </div>
        <div class='col-xs-3'></div>
    })
})
```

index.vash

```
@html.extend('layout', function(model)
{
    @html.block("body", function(model)
    {
        <div class=" col-xs-offset-3 col-xs-6 col-sm-6 col-md-6"
style="text-align: left;">
            <p><br>Welcome to the Fire Monitoring Assessment Platform
Portal, or FireMAP Portal.
            This is where all the work is done! You can upload imagery
to our databases,
            download imagery from our database, generate reports off of
these images,
            create training data with these images, and all you have is
create/login to your free account!
            If you have any questions or need help with anything,
please email dhamilton@nnu.edu. Thanks
            for using our service and remember... FireMap - Blazing the
trail for wildland fire research.</p>
```

```
        </div>
        <div class='col-xs-3'></div>
    })
})
```

layout.vash

```
@***This is the generic layout page for our html. All html that is the
same page to page will be contained here, as well as 'objects' that can
be overwritten for specific pages***@
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8"/>
        <title>@model.title</title> @***model means we'll pass in that
information in the controller for that specific page.***@
        <link href="/lib/bootstrap/dist/css/bootstrap.css"
rel="stylesheet"/>
        <link rel="stylesheet"
href="/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href="/lib/font-awesome/css/font-
awesome.css" />
        <link rel="stylesheet"
href="/lib/alertify/themes/alertify.core.css" />
        <link rel="stylesheet"
href="/lib/alertify/themes/alertify.default.css" />
        <link rel="stylesheet" href="/lib/farbtastic/farbtastic.css" />
        <link href="/css/site.css" rel="stylesheet"/>
        @html.block("cssFiles")@***Other pages will put their personal
CSS files in here as a variable if needed***@
    </head>
    <body>
        <div class="container-fluid">
            <div class = "row">

@***_________________________________Header___________________________
___________***@
                <div class="header nav navbar-static-top">
                    @if (model.user)
                    {
                        <div style="text-align:left"
class="usernamePosition">Logged in as @model.user</div>
                        <div style="text-align:right">
                            <form id="logoutForm" action="/Logout"
method="post" target="_self">
                                <button type="button"
class="logoutPosition" value = "Logout" name="logout"
onclick="logoutCall()">Logout</button>
                            </form>
                        </div>
                    }
                    @***Takes up the whole width of the screen.***@
                    <div class="col-xs-7 col-sm-offset-4 col-sm-4 col-
md-offset-4 col-md-4">
                        <h1>FireMAP</h1>
                        <h3>@model.pageHeader</h3>
                    </div>
                    <div class="col-xs-4 col-sm-4 col-md-4">
                        <img src="/Images/logo.png" alt="Northwest
Nazerene University Logo" class="logo" />
                    </div>
                </div>

@***__________________________________________________________________
___________***@

                @***__________________________Nav bar right below the
header______________________***@
                <div class="navbar-static-top">
                    @***Takes up the whole width of the screen.***@
                    <div class="col-md-12 shrinkText" id="nav">
                        <a href="/">About FireMAP</a>   |
                        <a href="image_uploader">Upload Imagery</a>   |
                        <a href="download_imagery">Download Imagery</a>
|
                        <a href="reports">Reports</a>   |
                        <a href="training_data_selector">Training Data
Selector</a>   |
                        <a href="login">Login</a>
                    </div>
                </div>

@***__________________________________________________________________
___________***@

                <div>@html.block("body")</div>@***Allows other pages to
be customized to what they need. More can be added if needed***@

                @***_____________________________Footer bar at the
bottom_________________________***@
                <div class="footer navbar navbar-fixed-bottom"
id="footer">
                    <div id="biggerFont"><a href="/">About</a></div>
                    <p></p>
                    <p>@model.footerAboutText</p>
                    <p> &copy 2017 - Northwest Nazerene University
Department of Math and Computer Science</p> @***&copy is throwing an
error, but it displays the copyright sign as it should***@
                </div>

@***__________________________________________________________________
___________***@
              </div>
          </div>

    @html.block("jsFiles") @***Other pages will put their JavaScript
files in here as a variable***@
    <script type="text/javascript" src="/js/layout.js"></script>
    </body>
</html>
```

login_page.vash

```
@html.extend('layout', function(model)
{
    @html.block("body", function(model)
    {
        <div class="row">
            <br>
            <br>
            <br>
            <br>
            <br>
            <br>
            <br>
                <p class="InvalidCreds">@model.invalidCredsText</p>
                <p class="logoutSuccess">@model.logoutSuccessText</p>

            <div class = "centerAlign">
                <form  id="LoginSubmit" class = "centerAlign"
action="/Login" method="post" target="_self">
                    <input type="text" id="username" name="username"
placeholder="Email" default="default" style = "width:200px"/>
                    <br>
                    <br>
                    <input type="password" id="userPassword"
name="password" placeholder = "Password" default="default" style =
"width:200px"/>
                    <br>
                    <span style="cursor: pointer; font-size: 11px;"
id="Help" onmouseover = "changeColor()" onmouseout = "changeBack()"
onclick="changeText()"> Forget Username or Password?</span>
                    <br>
                    <br>
                    <button type="button" value = "Create an Account"
name="CreateAccount" onclick="createAccountForm()">Create an
Account</button>
                    <button type="button" id ="Login" name = "Login"
onclick="validateForm()">Login</button>
                </form>

            </div>
        </div>
    })


    @html.block("jsFiles", function(model)
    {
        <script type="text/javascript"
src="/lib/alertify/alertify.min.js"></script>
        <script type="text/javascript" src="/js/login.js"></script>
    })

})
```

reports.vash

```
@html.extend('layout', function(model)
{
    @html.block("body", function(model)
    {
        <div class=" col-xs-offset-3 col-xs-6 col-sm-6 col-md-6"
style="text-align: center;">
            <p><br>This page is still being built. Thank you for your
patience.</p>
        </div>
        <div class='col-xs-3'></div>
    })
})
```

training_data_selector.vash

```
@html.extend('layout', function(model)
{
    @html.block("cssFiles", function(model)
    {

    })

    @html.block("body", function(model)
    {
        <div class="container-fluid">
            <div class="row">
                @***************************************************This
is where all the magic will
happen***********************************************************************
**************@
                <div class="main">
                    @*******************Tool
Section**********************************************************************
******************************************************@
                    <div class="col-xs-8 col-sm-8 col-md-2
toolSelector">
                        @****Save, Undo, and Delete buttons.****@
                        <div>
                            <button id="save" class="btn btn-success
submitButtons col-xs-12 col-sm-12 col-md-12 col-lg-4" type="button"
value="Save">Save <i class="fa fa-save"></i></button>
                            <button id="undo" class="btn btn-warning
submitButtons col-xs-12 col-sm-12 col-md-12 col-lg-4" type="button"
value="Undo">Undo <i class="fa fa-undo"></i></button>
                            <button id="redo" class="btn btn-primary
submitButtons col-xs-12 col-sm-12 col-md-12 col-lg-4" type="button"
value="Redo">Redo <i class="fa fa-repeat"></i></button>
                        </div>
                        @*****************************************@
                        @***Button don't line up super nicely....
Probably will need to create custom col sizes to better intregrate
buttons***@
                        <div class="selector">
                            Tool Selector
                            <br />
                            <div>
                                <button id="pointBtn" class="btn btn-
info btn-toolbar selectorButtonRow1 " type="button" value="Point">Point
<i class="fa fa-hand-pointer-o"></i></button>
```

```
                                <button id="pencilBtn" class="btn btn-
info btn-toolbar selectorButtonRow1 colorNotSelected" type="button"
value="Pencil">Pencil <i class="fa fa-pencil"></i></button>
                            </div>
                            <div>
                            <button id="lineBtn" class="btn btn-info
btn-toolbar selectorButtonRow1 colorNotSelected" type="button"
value="Line">Line <i class="fa fa-ellipsis-h"></i></button>
                            <button id="polylineBtn" class="btn btn-
info btn-toolbar selectorButtonRow1 colorNotSelected" type="button"
value="Polyline">PolyLine <i class="fa fa-bolt"></i></button>
                            </div>
                            <div class="selectorButtonRow2">
                                <button id="polygonBtn" class="btn btn-
info btn-toolbar selectorButtonRow2 colorNotSelected" type="button"
value="Polygon">Polygon <i class="fa fa-star"></i></button>
                                <button id="circleBtn" class="btn btn-
info btn-toolbar selectorButtonRow2 colorNotSelected" type="button"
value="Circle">Circle <i class="fa fa-circle-o"></i></button>
                            </div>
                            <div class="selectorButtonRow3">
                                <button id="auto_clusterBtn" class="btn
btn-info btn-toolbar selectorButtonRow3 colorNotSelected" type="button"
value="Auto-Cluster">Auto-Cluster <i class="fa fa-
automobile"></i></button>
                                <button id="flood_fillBtn" class="btn
btn-info btn-toolbar selectorButtonRow3 colorNotSelected" type="button"
value="Flood Fill">Flood Fill <i class="fa fa-tint"></i></button>
                            </div>
                        </div>

@***********************************************************************
****************************************************@
                        @***********Label, Not perfect, but it works
pretty well*************@
                        <div>
                            <form id="labelForm"><input
class="dataLabel col-xs-8 col-sm-8 col-md-7" id="label"
placeholder="Input Label Here"></form>
                            <span id="labelOptionsTxt" class="text-
muted infoText hide">Label Options:</span>
                            <select class="col-xs-8 col-sm-8 col-md-7
dataLabel hide" name="labelDropDown" id="labelDropDown"></select>
                            <button id="btnLabel" class="btn btn-sm
btn-success submitButtons labelWidth col-xs-8 col-sm-8 col-md-5"
type="button" value="Label">Label <i class="fa fa-check"></i></button>
                        </div>

@*********************************************************************@


                        <div>
                            <span class="text-muted
infoText">Showing:</span>
                            <select class="col-md-12 labelSelect"
name="label" id="dropDown">
                                <option value="All Labels">All
Labels</option>
```

```
                            </select>
                        </div>
                        @***Current Pizel Coordinates and Dimensions of
Photo, off by a couple of picles on the right side, not a big deal
though***@
                        <div>
                            <div class="col-md-6"
id="noPaddingRightMarginSpace">
                                <input id="coordinates" type="text"
class="pixelLocations" value="(0,0)" readonly />
                                <p class="text-muted
infoText">Coordinates(x,y)</p>
                            </div>
                            <div class="col-md-6"
id="noPaddingLeftMarginSpace">
                                <input id="dimensions" type="text"
class="pixelLocations" value="(0,0)" readonly />
                                <p class="text-muted
infoText">Dimensions(x,y)</p>
                            </div>
                        </div>

@**********************************************************************
*****************************************************@

@*************************Zoom*******************@
                        <div class="zoom col-xs-12 col-sm-12 col-md-
12">
                            <div class="zoomIcon floatLeft"><button
id="scaleOut" class="btn" type="button"><i class="fa fa-search-
minus"></i></button></div>
                            <div class="zoom col-xs-9 col-sm-9 col-md-
9" style="float: left">
                                <input type="range" id="size" min="100"
step="10">
                            </div>
                            <div class="zoomIcon col-md-offset-11
clearRight"><button id="scaleIn" class="btn" type="button"><i class="fa
fa-search-plus"></i></button></div>
                            <div class="col-xs-12"><p id="resizeText"
class="text-muted infoText" style="text-align: center;">Shift-Scroll to
resize Image.</p></div> @***Make sure the step here matches the step in
the js file***@
                        </div>

@**************************************************@

@**********************************************************************
*********************************************************************@
                    </div>

@**************************************************Image
Section*****************************************************************@
                    <div class="col-xs-12 col-sm-12 col-md-8">
                        <div class="trainingDataSection">
                            @***Buttons***@
                            <div id="hideOriginalUploadButton">
```

```
                                <button id="tutorial" class="btn btn-lg
btn-default submitButtons col-md-offset-1" type="button"
value="Tutorial">Turorial <i class="fa fa-question-circle-o"></i>
</button>
                                <input id="upload" type="file"
class="noShow" /> @**Button tricked into being an input file idea came
from http://stackoverflow.com/questions/11406605/how-to-make-a-link-
act-as-a-file-input *****@
                                <button id="uploadButton" class="btn
btn-lg btn-info submitButtons col-md-offset-4" type="button"
value="Select Training Image">Select Training Image <i class="fa fa-
file-image-o"></i> </button>
                                <button id="attributeTable-btn"
class="btn btn-lg btn-info submitButtons " type="button" value="Select
Attribute Table">Select Attribute Table  <i class="fa fa-
table"></i></button>
                                <input id="attributeTable" type="file"
class="noShow" /> @****Button tricked into being an input file idea
came from http://stackoverflow.com/questions/11406605/how-to-make-a-
link-act-as-a-file-input ****@
                            </div>
                            @************@
                            @***Image***@
                            <div id="image">
                                <div id="scaledImage"
class="dragscroll" tabindex="">
                                    <div id="spinner"><i class="fa fa-
spinner fa-spin fa-4x"></i></div>
                                    <div id="canvasAndImageDiv"
class="canvasAndImageDiv">
                                        <div class="shapeLabelToolTip"
id="shapeToolTip"><p>Shape Label Text</p></div>
                                        <canvas id="myCanvas"
class="grabCursor" width="1" height="1"></canvas>
                                    </div>
                                </div>
                            </div>
                            @***********@
                        </div>
                    </div>

@***********************************************************************
*****************************************************@
                    @************************Color Picker
Section*******************************@
                    <div class="col-xs-offset-8 col-sm-offset-8 col-md-
offset-0 col-md-2 colorPicker">
                        <button id="delete-btn" class="btn btn-danger
marginSpace loadAndDeleteButtons rightMargin col-sm-6 col-md-6 col-lg-
6" type="button" value="Confirm">Delete  <i class="fa fa-
close"></i></button>
                        <button id="load-btn" class="btn btn-info
marginSpace loadAndDeleteButtons leftMargin col-xs-6 col-sm-6 col-md-6
col-lg-6" type="button" value="Confirm">Load  <i class="fa fa-folder-
open-o"></i></button>
                        <input id="load" type="file" class="noShow" />
@***Button tricked into being an input file idea came from
```

```
http://stackoverflow.com/questions/11406605/how-to-make-a-link-act-as-
a-file-input ****@
                        @***Will confirm the placement of a polyline,
allows user to create different polylines without having to click off
the polyline button***@
                        <button id="polylineConfirmation" class="btn
btn-success marginSpace submitButtons hide  col-xs-12 col-sm-12 col-md-
12 col-lg-12" type="button" value="Confirm">Confirm Polyline <i
class="fa fa-check"></i></button>
                        <button id="completePolygon" class="btn btn-
success marginSpace submitButtons hide col-xs-12 col-sm-12 col-md-12
col-lg-12" type="button" value="Confirm">Auto-Complete Polygon <i
class="fa fa-check"></i></button>
                        <div class="colorPickerBackground">
                            <span class="col-lg-offset-4">Color
Picker</span>
                            <br />
                            <div class="col-lg-offset-1">
                                <div class="colorRow">
                                    <button id="btn-black" class="btn
btn-lg btn-black marginSpace" type="button" value="Black"></button>
                                    <button id="btn-white" class="btn
btn-lg btn-white marginSpace" type="button" value="White"></button>
                                    <button id="btn-darkRed" class="btn
btn-lg btn-darkRed marginSpace" type="button" value="Dark
Red"></button>
                                    <button id="btn-red" class="btn
btn-lg btn-red marginSpace" type="button" value="Red"></button>
                                </div>

                                <div class="colorRow">
                                    <button id="btn-orange" class="btn
btn-lg btn-orange marginSpace" type="button" value="Orange"></button>
                                    <button id="btn-yellow" class="btn
btn-lg btn-yellow marginSpace" type="button" value="Yellow"></button>
                                    <button id="btn-neonGreen"
class="btn btn-lg btn-neonGreen marginSpace" type="button" value="Neon
Green"></button>
                                    <button id="btn-pukeGreen"
class="btn btn-lg btn-pukeGreen marginSpace" type="button" value="Puke
Green"></button>
                                </div>

                                <div class="colorRow">
                                    <button id="btn-lightBlue"
class="btn btn-lg btn-lightBlue marginSpace" type="button" value="Light
Blue"></button>
                                    <button id="btn-blue" class="btn
btn-lg btn-blue marginSpace" type="button" value="Blue"></button>
                                    <button id="btn-purple" class="btn
btn-lg btn-purple marginSpace" type="button" value="Purple"></button>
                                    <button id="btn-pink" class="btn
btn-lg btn-pink marginSpace" type="button" value="Pink"></button>
                                </div>

                                <div class="colorRow">
```

```
                                    <button id="btn-user1" class="btn
btn-lg btn-white marginSpace btn-user1" type="button"
value="White"></button>
                                    <button id="btn-user2" class="btn
btn-lg btn-white marginSpace btn-user2" type="button"
value="White"></button>
                                    <button id="btn-user3" class="btn
btn-lg btn-white marginSpace btn-user3" type="button"
value="White"></button>
                                    <button id="btn-user4" class="btn
btn-lg btn-white marginSpace btn-user4" type="button"
value="White"></button>
                                </div>

                            </div>

                            <div class="userColorChoice">
                                <div class="colorWheelPicker col-xs-
offset-0">
                                    <form id="colorWheelForm">
                                        <input type="text" id="color"
value="#123456" />
                                        <button type="button"
id="colorPickerBtn" class="btn btn-sm btn-success selectorButtonRow1"
value="Assign Color">Assign Color <i class="fa fa-paint-brush"></i>
</button>
                                    </form>
                                </div>
                                <div id="colorwheelpicker">
                                    <div id="colorpicker"></div>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div id="changeUploadButtonPosition" class="col-xs-
12 col-sm-12 col-md-12 col-lg-2">
                        <button id="tutorial2" class="btn btn-lg btn-
default movedUploadedButtons col-xs-12 col-sm-12 col-md-12"
type="button" value="Tutorial">Turorial <i class="fa fa-question-
circle-o"></i> </button>
                        <input id="upload2" class="noShow" type="file"
/> @***Button tricked into being an input file idea came from
http://stackoverflow.com/questions/11406605/how-to-make-a-link-act-as-
a-file-input ****@
                        <button id="uploadButton2" class="btn btn-lg
btn-info movedUploadedButtons movedUploadButton col-sm-12 col-md-12"
type="button" value="Select Training Image">Select Training Image <i
class="fa fa-file-image-o"></i> </button>
                        <button id="attributeTable-btn2" class="btn
btn-lg btn-info submitButtons col-sm-12 col-md-12" type="button"
value="Select Attribute Table">Select Attribute Table  <i class="fa fa-
table"></i></button>
                        <input id="attributeTable2" type="file"
class="noShow" /> @****Button tricked into being an input file idea
came from http://stackoverflow.com/questions/11406605/how-to-make-a-
link-act-as-a-file-input *****@
                     </div>
```

```
@*******************************************************************
*********@
                </div>

@*************************************************************Popup
that covers whole screen when user wants to save their
data******************************************@
                <div id="popupFormHolder" style = "overflow:hidden">
                    <div id="popupForm">
                        <form  id="saveSubmit" action="/trainingData"
method="post" target="_blank">
                            <div id="sliderValue" class="col-xs-12">
                                <p style="width: 100%"> Percent to be
used for testing purposes (0 - 100): </p>
                                <input id="percentTesting" type="range"
min="0" step="1" max ="100" defaultvalue="30" value="30" class="show
col-xs-5" name ="percentTesting" oninput="percentTestingValue.value =
percentTesting.valueAsNumber+'%' "/>
                                <output id="percentTestingValue"
name="percentTestingValue" for="range" class="col-xs-offset-
1">30%</output>
                            </div>
                            <div id ="sliderValue" class="col-xs-12">
                                <p class="col-xs-12">Pixel Gap to be
used (Smaller values means more data):</p>
                                <input id="pixelGap" type="range"
min="0" step="1" max="250" defaultvalue="5" value="5" class="show col-
xs-5" name ="pixelGap" oninput="pixelGapValue.value = 'Pixel Gap: ' +
pixelGap.valueAsNumber"/>
                                <output id="pixelGapValue"
name="pixelGapValue" for="range" class="col-xs-3">Pixel Gap: 5</output>
                            </div>
                            <div id="sliderValue" class="col-xs-12">
                                <p class="col-xs-12"> Delete duplicate
data points between training and testing files:</p>
                                <input id="deleteDuplicatesValue1"
type="radio" style="vertical-align: middle; margin: 0px;" name
="deleteDuplicates" value ="1" checked/> Yes<br>
                                <input id="deleteDuplicatesValue2"
type="radio" style="vertical-align: middle; margin: 0px;" name
="deleteDuplicates" value ="2"/> No<br>
                            </div>
                            <div id="sliderValue" class="col-xs-12">
                                <p class="col-xs-12"> Download canvas
training image with drawings:</p>
                                <input id="downloadCanvasValue1"
type="radio" style="vertical-align: middle; margin: 0px;" name
="downloadCanvas" value ="1" checked/> Yes<br>
                                <input id="downloadCanvasValue2"
type="radio" style="vertical-align: middle; margin: 0px;" name
="downloadCanvas" value ="2"/> No<br>
                            </div>
                            <div id="formSubmit" class="col-xs-12">
                                <br>
                                <br>
                                <button id="formButtons" type="button"
onclick="validateForm()" class="btn btn-lrg btn-success
submitButtons">Submit</button>
                                <button id="formButtons" type="button"
onclick="cancelForm()" class="btn btn-lrg btn-danger
submitButtons">Cancel</button>
                                <br>
                                <input id="serverData" type="text"
class="hide" name ="drawingData" value=""/>
                                <input id="serverDataFileName"
type="text" class="hide" name ="drawingDataFileName" value=""/>
                                <br>
                                <p id ="submtInfo">Once you hit submit,
a new window will open. This window is generating your data.
                                It will close automatically once it is
done writing your files.
                                If you close the window prematurely it
will corrupt your data.</p>
                            </div>
                        </form>
                    </div>
                </div>

@***********************************************************************
************************************************************************
***************************@
            </div>
        </div>
    })

    @html.block("jsFiles", function(model)
    {
        <script type="text/javascript"
src="/lib/jquery/dist/jquery.js"></script>
        <script type="text/javascript"
src="/lib/bootstrap/dist/js/bootstrap.js"></script>
        <script type="text/javascript"
src="/lib/_app/dragscroll.js"></script>@***Minified version - Allows
mouse drag to count as a scroll on our image ***@
        <script type="text/javascript"
src="/lib/_app/FileSaver.js"></script>@***Minified version***@
        <script type="text/javascript"
src="/lib/underscore/underscore.js"></script>
        <script type="text/javascript"
src="/lib/_app/easel.js"></script> @***Minified version***@
        <script type="text/javascript"
src="/lib/alertify/alertify.min.js"></script>
        <script type="text/javascript"
src="/lib/farbtastic/farbtastic.js"></script>
        <script type="text/javascript" src="/lib/_app/canvas-
toBlob.js"></script>
        <script type="text/javascript"
src="/js/training_data_selector.js"></script>

        @****<script type="text/javascript"
src="/lib/_app/training_data_selector.js"></script> Minified js, will
want to use non-minified until published.****@
```

```
    })
})
```

upload_imagery.vash

```
@html.extend('layout', function(model)
{
    @html.block("body", function(model)
    {
        <div class=" col-xs-offset-3 col-xs-6 col-sm-6 col-md-6"
style="text-align: center;">
            <p><br>This page is still being built. Thank you for your
patience.</p>
        </div>
        <div class='col-xs-3'></div>
    })
})
```

README.md

~~~
**Author's Note**

Welcome to the documentation of the FireMAP Portal! As the portal is
incomplete the documentation is also incomplete. If you are reading
this
because you are working on the FireMAP Portal then I suggest you add
documentation as you go and not later. Doing so will make your life a
lot easier.

The documentation follows a generic pattern. The beginning talks about
the
general layout of the project and where subsequent pages will talk
about
each file that wasn't auto-generated or someone else's library. Any
links leading to outside documentation are working at the time of this
document being written but I cannot guarantee they will work when you
need them. If that is the case then I suggest searching the library I
used to find their new documentation page.

**Local Development**
Local development should be done using Vagrant. This allows you to
create
a virtual machine that is similar to the server that hosts the firemap-
portal
in production. This process should work on Windows, MacOS and Linux.
Install

* VirtualBox (https://www.virtualbox.org/wiki/Downloads)
* Vagrant (https://www.vagrantup.com/downloads.html)
* Vagrant plugins

```
vagrant plugin install vagrant-vbguest
vagrant plugin install vagrant-hostmanager
```

***Start local virtual machine***

```
vagrant up
```

***Restart local virtual machine***

```
vagrant reload
```

***Shutdown local virtual machine***

*This will delete any data changes made in the database

```
vagrant destroy --force
~~~

~~~~
```

**General Code Layout**

The FireMAP portal follows the Model-View-Controller (MVC) layout.

The views are .vash views located in the "Views" folder. A .vash file
is
essentially an html file that allows the programmer to put variables
inside html. There will be more on how that works later in the
"Views/.vash" documentation. It should be noted that in this project,
views refer to only the .vash files, and nothing else, including CSS.

The controllers are in the "Controllers" folder. It should be noted
that
in this project, controllers refer to server side JavaScript only. They
do not refer to the client side JavaScript. There will be more specific
details about the controllers in the "Controllers/Server Side
JavaScript" section.

The Model of the FireMAP Portal is to create a centralized location for
all things FireMAP. That's a very broad goal so let's break it down
some.

-   The "About FireMAP" page (aka the Home page) is to inform users
what
    the FireMAP portal is and what FireMAP is.

-   The "Upload Imagery" page is to allow users the capability to
upload
    their data to the FireMAP database. This would allow them to
    store/share their data with us and run our classifiers on their
    data.

-   The "Download Imagery" page is to allow users to browse our
    database/server for images to use or to find the ones they've
    uploaded.

-   The "Reports" page is to allow users to download/view pre-generated
    reports. These reports would be the ones that our classifiers
    created and are being stored on the database/server.

-   The "Training Data Selector" is to allow users to upload an image
    into their browser (not our database/server) to draw on, label, and
    save data from. It must provide the flexibility and accuracy needed
    to create accurate training data for the user.

-   The "Login" page is to allow users to access any page but the
"About
    FireMAP" page (as that one is public). A user will not be able to
    interact with the "Upload Imagery", "Download Imagery", "Reports",
    or "Training Data Selector" page without first logging in.

**server.js**

The Server.js file is the backbone of this project. You can think of it
as the "main" function of a C++ program. The constants are npm (node
~~~~

package manager -\> JavaScript Libraries) that are pulled in and then used as objects (if you think of object orientated programming). This is
where the connection to the FireMAP database is made and it is where the
port the portal is listening for is set. It also sets up our server side
controllers.

controllers/Server Side JavaScript

In this section, we talk about all of the JavaScript files in the "Controllers" folder.

- index.js
    - This file is simple, it sets all the other controllers to their corresponding web page.
- home_controller.js
    - This file grabs the Index.vash page and adds the variables the Index.vash file is looking for. This controller allows the Home page to be labeled the Home page.
- upload_imagery_controller.js
    - Does nothing
- download_imagery_controller.js
    - Does nothing
- reports_controller.js
    - Does nothing
- training_data_selector_controller.js
    - This file does a lot. The first thing it does is adds the variables the TrainingDataSelector.vash file is looking for.
    - When the user hits submit on the save button on this page, that data is then controlled here.
        - First it creates a file path for the Training csv file to be stored in
        - Next it will run the CoordinateSelector C++ program with the Training csv and the inputs given to it from the save form.
        - Then it will save these two csv files with the Training csv file.

        - Finally it will zip these three csv files into 1 zip folder and download that zip for the user.

- logout_controller.js

    - This kills the session that was created upon login, effectively logging the user out

That wraps it up for the "Controllers" folder. If you add a new page to the portal then a new Controller will need to be created for that page. Any information that is sent to the server will be handled through the controller.

views/.vash

In this section, we talk about the "Views" folder or the .vash files. .vash files are just an html file that allows the programmer to create sections (variables) inside their html that can then be page specific.

- layout.vash

    - This is the most important .vash file because it controls the layout for every other page in the portal. This is where the header, nav bar, and footer are set for each page. This is how to maintain a common look among all pages.

    - The parts where you can change to make each page specific is located with "\@html.block". You will see this code in the other .vash files, anything in there is specific to that page. To create a spot that will be specific per page that isn't already defined. Look at the "body" block or the "jsFiles" block in this file.

- index.vash

    - This is the .vash file that contains the html for the Home/About page. All it does it change the body of the page to include a paragraph of text explaining the FireMAP portal and FireMAP

- upload_imagery.vash

    - Does nothing

- download_imagery.vash

    - Does nothing

- reports.vash

    - Does nothing

- training_data_selector.vash

    - This is the .vash file that contrains the html for the

TrainingDataSelector page. It changes the body accordingly and adds some JavaScript for validating user input.

- I would suggest reading the comments for this page to best understand it as it breaks it up into chunks, but I'll go over somethings here.

- The Tool Section contains all the buttons for the drawing tools,
save, undo, redo, zoom in, zoom out, and label buttons, and the zoom slider. It also contains the label textbox (or dropdown menu if you're using the Attribute Table), the coordinates of your current mouse, and the dimension of the image.

- The Image Section contains the Tutorial, Select Training Image, and Select Attribute Table buttons until the user selects an image. Then only the image is present in this section.

- The Color Picker Section includes the Delete and Load buttons, as well as the color buttons and the color picker wheel. After a
user has selected an image to draw on it will contain the Select
Training Image, the Select Attribute Table, and the Tutorial buttons as well.

- login_page.vash

    - This is the .vash file that contains the html for the Login page. It changes the body accordingly and adds some JavaScript files for validating user input.

- create_account.vash
    - This is the .vash page for the create account page. It is a form that takes the user information and passes it to the auth/index.js script.

JavaScript

In this section, we talk about the "JS" folder inside the "Public" folder.

- Canvas-toBlob.js

    - This isn't a JavaScript file that I wrote. It takes the canvas drawing that the user does on the TrainingDataSelector page and converts it to a blob object that can then be downloaded as a jpeg.

- download_form.js

    - This isn't a JavaScript file that I wrote and I believe it is a file that is no longer being used. But because I don't want to remove it and then find out months down the road that it was needed somewhere.

- dragscroll.js

    - This isn't a JavaScript file that I wrote. It allows the canvas on the TrainingDataSelector page to be scrolled/moved around by clicking and dragging on the canvas.

- easel.js

    - This isn't a JavaScript file that I wrote. It works in conjunction with CreateJS. Without this file drawing on the canvas would not work.

- FileSaver.js

    - This isn't a JavaScript file that I wrote. Helps save the canvas and the csv files and send them as a download. Because node.js now handles csv saving, this is primarily used to save the canvas as a jpeg.

- login.js

    - This is the JavaScript for the login page. It validates the form fields and then sends that data to the server.

- create_account.js

    - This is the JavaScript for the Create Account page. It validates the form fields and then sends that data to the server.

- training_data_selector.js

    - This is the JavaScript file that is the bread and butter of this application. It is very long and so I'll break it up into pieces.

    - The beginning of the file contains constants, JavaScript objects, and event listeners.

    - Once an image is loaded in the rest of the buttons will have event listeners attached to them ("hasDrawn()").

    - The functions inside of this file are well documented. Because of this, I will leave out their documentation from this file.

lib

- The files located in the "Lib" folder inside the "Public" folder contains code from pulled in libraries.

images

- The files located in the "Images" folder inside the "Public" folder contains static images that are used on different pages throughout the website.

```
**css**

-   The files located in the "css" folder inside the "public" folder
    contain the style of the web site.

**Server Modules**

-   The files located in the "ServerModules" folder is code that runs
on
    Node.js and do things server side.

    -   log.js

        -   This file contains logging code that makes it easier to log
            information to view for debugging and other reasons.

**data/data scripts**

-   This folder contains the scripts that handle any database calls or
    functions that the portal uses.

    -   database.js

        -   This file contains the connection parameters to the MySQL
            database as well as functions that other scripts would
            use when interacting with the database. Need something
            done through the database? Do it here.

    -   index.js

        -   This script just initializes the database.js script.
            If there was more scripts in the data folder it would
            initialize them too.

**auth/authorization scripts**

-   This folder contains the scripts that authorize the user.

    -   hasher.js

        -   This is the script that will compute the user's
            hash and salt on their password when they create
            an account and when they login.

    -   index.js

        -   This is the script that handles the post of login
            and create account.

**coordinate_selector**

-   This folder contains the FireMAPCoordinateSelector executable
    and holds any training data zips that are created by the user.

**config.js**
```

```
-   This script contains configuration information for the server,
    such as the database port, user, and password.
    Keep it secret - keep it safe.

**GruntFile.js**

-   This script allows the user to just reload the page and
    see their changes without having to restart node. Very handy for
development.

***Instructions to run on local machine***

-   If you want to run the Training Data Selector offline then you can
do the
    following (this works as of 4/16/2018). Future work may change how
this is done.

    -   Download Node.js from https://nodejs.org/en/download/ and
install.

    -   Download the Training Data Selector zip folder.

    -   Download MySQL database and set up a database using the MySQL
Dump
        (bootstrap.sql) in the Training Data Selector zip folder. Make
sure
        this database is up and running on your computer.

    -   Open up the database.js file in the data folder and make
        the line reading "//password: config.database.password,"
        uncommented and then save.

    -   Unzip and open the Training Data Selector folder and go to the
location
        of the "server.js" file. Once that location is found
        (Ex. C:\Users\Me\Desktop\Portal\portal)
        go to the location via the terminal window (command prompt).
        Once in that folder's directory, simply type
        "node server.js" without the quotes and then go to
        localhost:1337. The Training Data Selector will be hosted
there.

    -   If you are having issues with the FireMAPCoordinateSelector,
        then you may want to ensure you have the correct version.
        The default version is a .out executable as the server us a
Linux machine.
        However, Windows computers cannot read .out files. If this is
the case, you
        will need to download the Windows version of the
FireMAPCoordinateSelector
        and replace the .out file with the .exe file.
```