Tripal D3.js Documentation

LA Sanderson et. al., University of Saskatchewan

Aug 31, 2022

Contents:

Provides d3.js integration for Tripal. It provides an API for developing consistent biological diagrams with a common configuration, as well as, providing some common diagrams such as pie, bar, column and pedigree diagrams.

NOTE: This module is an API and does not provide user facing diagrams. Rather, you would use this API to easily draw common diagrams within your own fields. For an example of how to use this module, see Tripal Fancy Fields.

CHAPTER 1

Installation

1.1 Quickstart

1. Unpack the D3 v3 javascript library in your Drupal Libraries directory (quick check, you should have a libraries/d3/d3.min.js file; for more information see the drupal.org documentation).
2. Download and install this module as you would any other Drupal module (Documentation)
3. (Optional) Go to Admin » Tripal » Extension Modules » Tripal D3 Diagrams to configure colour schemes, etc.

1.2 Dependencies

1. Tripal 3.x
2. Drupal Libraries API
3. D3 v3 javascript library

1.3 Installation

This installation guide assumes you already have a Tripal site. If not, see the Tripal installation documentation.

1. Install the Drupal libraries API

```
drush pm-download libraries
```

2. Install this module.

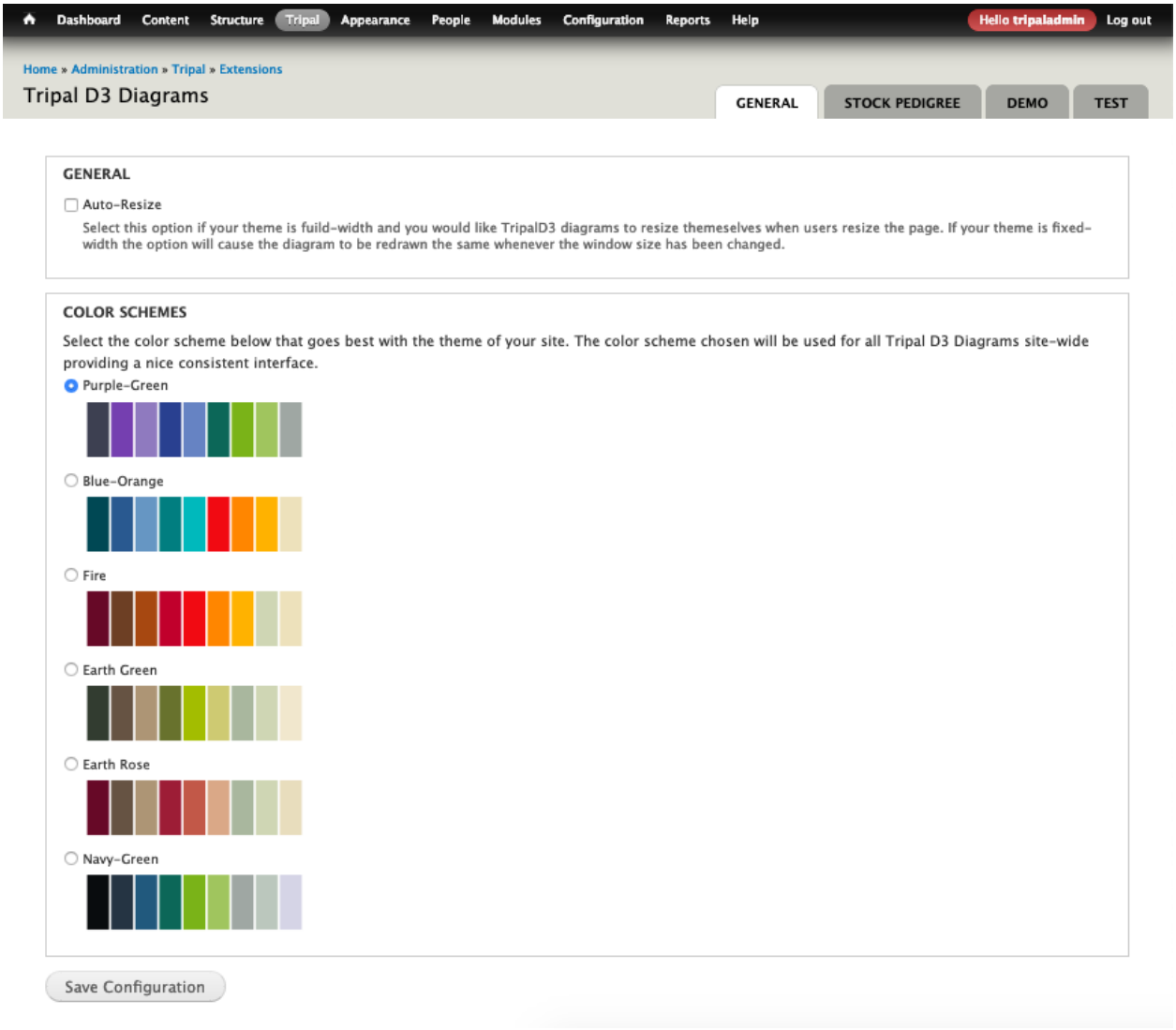
```
git clone https://github.com/tripal/tripald3.git
drush pm-enable tripald3
```

Now that the module is installed, you can configure it to suit your needs or just get back to developing your custom field with easy to add diagrams.

CHAPTER 2

Configuration

Configuration can be found at Admin » Tripal » Extension Modules » Tripal D3 Diagrams. Configuration has been kept simple to lower the administrative needs of this module.



As you can see from the above screenshot, the main configuration options are related to the colour scheme of the diagrams. This is one of the main features of the API which allows your diagrams to look professional by being consistent in colour scheme across your site. Simply choose on of the preset colour schemes or add your own *Custom Colour Schemes*.

CHAPTER 3

Developers Guide

This guide contains tutorials for using the Tripal D3.js API to add diagrams to your modules.

3.1 Draw a Simple Pie Chart

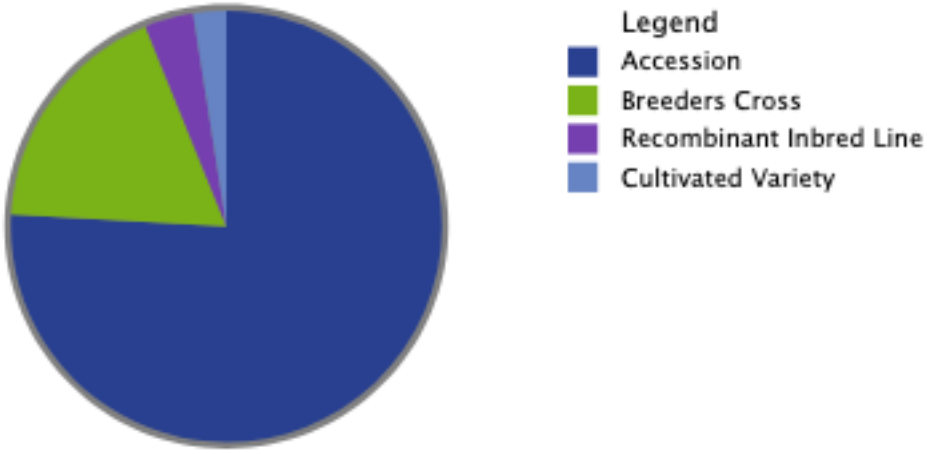


Figure: Proportion of Tripalus databasica Germplasm Types. *The above pie chart depicts the ratio of germplasm types available for Tripalus databasica.*

1. Load the API into the page you would like your diagram using `<php tripald3_load_libraries();?>`
2. Retrieve you data and manipulate it into the structure required by the chart. This can be done a number of ways, the easiest of which is to query your database in your Drupal preprocess hook and then save the results as a javascript setting.

```
/**
 * Preprocess hook for template my_example.tpl.php
 * The module name is demo.
 */
function demo_my_example_preprocess(&$variables) {

  // Load the API (Step #1 above)
  tripald3_load_libraries();

  // Retrieve your data.
  // For this example  we're just going to define the array directly.
  $data  = [
    [
      "label": "Accession",
      "count": 2390,
    ],
    [
      "label": "Breeders Cross",
      "count": 567,
```

(continues on next page)

(continued from previous page)

```
    ],
    [
      "label": "Recombinant Inbred Line",
      "count": 115,
    ],
    [
      "label": "Cultivated Variety",
      "count": 78,
    ],
  ];

  // Make it available to javascript via settings.
  $settings = array(
    // Always namespace to your module to avoid collisions.
    'demo' => array(
      // Pass in your data using a descriptive settings key.
      'stockTypePieData' => $data,
    ),
  );
  drupal_add_js($settings, 'setting');
}
```

3. Add a container element in your template where you would like the chart drawn.

```
<div id="tripald3-simplepie" class="tripald3-diagram">
  <!-- Javascript will add the Simple Pie Chart, Title and Figure legend here -->
</div>
```

4. Draw the chart in your template by calling *tripalD3.drawChart()*. This is done within a script tag using Drupal behaviours to ensure it is run at the correct point and the data prepared is passed in.

```
<script type="text/javascript">
  Drupal.behaviors.tripalD3demoSimplePie = {
    attach: function (context, settings) {

      // Pull the data out of the javascript settings.
      var data = Drupal.settings.demo.stockTypePieData;

      // Draw your chart.
      tripalD3.drawFigure(
        data,
        {
          "chartType" : "simplepie",
          "elementId": "tripald3-simplepie",
          "height": 250,
          "width": 500,
          "keyPosition": "right",
          "title": "Proportion of <em>Tripalus databasica</em> Germplasm Types",
          "legend": "The above pie chart depicts the ratio of germplasm types␣
↪available for <em>Tripalus databasica</em>.",
        }
      );
    }
  };
</script>
```

5. There is no step #5; you're done!

3.2 Draw a Simple Donut Chart

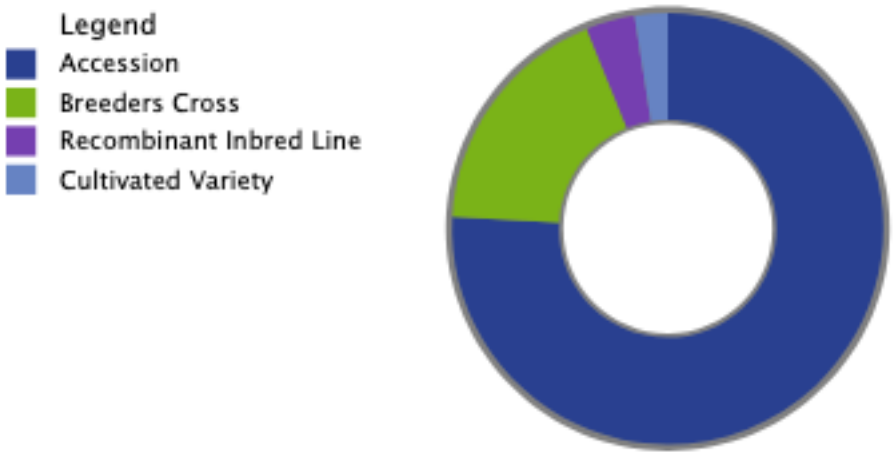


Figure: Proportion of Tripalus databasica Germplasm Types. *The above pie chart depicts the ratio of germplasm types available for Tripalus databasica.*

1. Load the API into the page you would like your diagram using `<php tripald3_load_libraries();?>`
2. Retrieve you data and manipulate it into the structure required by the chart. This can be done a number of ways, the easiest of which is to query your database in your Drupal preprocess hook and then save the results as a javascript setting.

```
/**
 * Preprocess hook for template my_example.tpl.php
 * The module name is demo.
 */
function demo_my_example_preprocess(&$variables) {

  // Load the API (Step #1 above)
  tripald3_load_libraries();

  // Retrieve your data.
  // For this example  we're just going to define the array directly.
  $data  = [
    [
      "label": "Accession",
      "count": 2390,
    ],
    [
      "label": "Breeders Cross",
      "count": 567,
    ],
    [
      "label": "Recombinant Inbred Line",
```

(continues on next page)

(continued from previous page)

```
      "count": 115,
    ],
    [
      "label": "Cultivated Variety",
      "count": 78,
    ],
  ];

  // Make it available to javascript via settings.
  $settings = array(
    // Always namespace to your module to avoid collisions.
    'demo' => array(
      // Pass in your data using a descriptive settings key.
      'stockTypeDonutData' => $data,
    ),
  );
  drupal_add_js($settings, 'setting');
}
```

3. Add a container element in your template where you would like the chart drawn.

```
<div id="tripald3-simpledonut" class="tripald3-diagram">
  <!-- Javascript will add the Simple Donut Chart, Title and Figure legend here -->
</div>
```

4. Draw the chart in your template by calling *tripalD3.drawChart()*. This is done within a script tag using Drupal behaviours to ensure it is run at the correct point and the data prepared is passed in.

```
<script type="text/javascript">
  Drupal.behaviors.tripalD3demoSimpleDonut = {
    attach: function (context, settings) {

      // Pull the data out of the javascript settings.
      var data = Drupal.settings.demo.stockTypeDonutData;

      // Draw your chart.
      tripalD3.drawFigure(
        data,
        {
          "chartType" : "simpledonut",
          "elementId": "tripald3-simpledonut",
          "height": 250,
          "width": 500,
          "keyPosition": "left",
          "title": "Proportion of <em>Tripalus databasica</em> Germplasm Types",
          "legend": "The above donut chart depicts the ratio of germplasm types␣
↪available for <em>Tripalus databasica</em>.",
        }
      );
    }
  };
</script>
```

5. There is no step #5; you're done!

3.3 Draw a Multi-series Donut

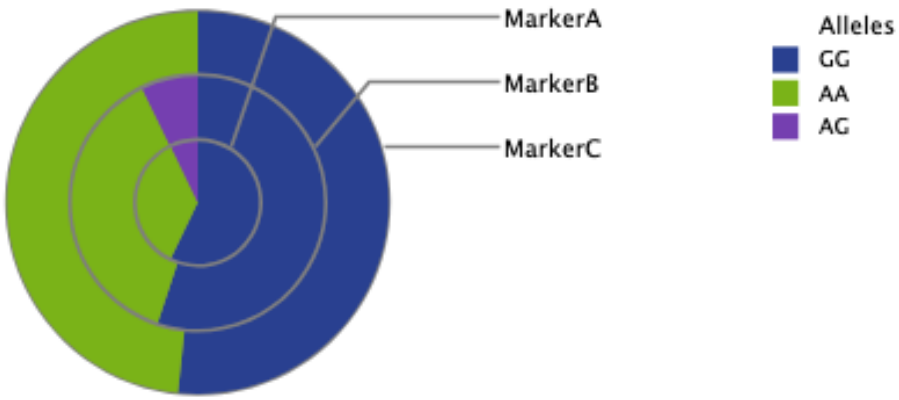


Figure: Comparison of allele calls across 3 FBA-1 markers. *The above chart shows the allele ratios for three seperate markers assaying the FBA-1 (fictional but amazing) gene.*

1. Load the API into the page you would like your diagram using `<php tripald3_load_libraries();?>`
2. Retrieve you data and manipulate it into the structure required by the chart. This can be done a number of ways, the easiest of which is to query your database in your Drupal preprocess hook and then save the results as a javascript setting.

```
/**
 * Preprocess hook for template my_example.tpl.php
 * The module name is demo.
 */
function demo_my_example_preprocess(&$variables) {

  // Load the API (Step #1 above)
  tripald3_load_libraries();

  // Retrieve your data.
  // For this example  we're just going to define the array directly.
  $multiDonutData = [
    [
      "label": "MarkerA",
      "parts": [
        [
          "label": "GG",
          "count": 16,
        ],
        [
          "label": "AA",
          "count": 10,
        ],
        [
          "label": "AG",
          "count": 2,
        ],
```

(continues on next page)

(continued from previous page)

```
      ],
    ],
    [
      "label": "MarkerB",
      "parts": [
        [
          "label": "GG",
          "count": 145,
        ],
        [
          "label": "AA",
          "count": 99,
        ],
        [
          "label": "AG",
          "count": 19,
        ],
      ],
    ],
    [
      "label": "MarkerC",
      "parts": [
        [
          "label": "GG",
          "count": 78,
        ],
        [
          "label": "AA",
          "count": 73,
        ],
      ],
    ],
  ];

  // Make it available to javascript via settings.
  $settings = array(
    // Always namespace to your module to avoid collisions.
    'demo' => array(
      // Pass in your data using a descriptive settings key.
      'stockTypeDonutData' => $multiDonutData,
    ),
  );
  drupal_add_js($settings, 'setting');
}
```

3. Add a container element in your template where you would like the chart drawn.

```
<div id="tripald3-multidonut" class="tripald3-diagram">
  <!-- Javascript will add the Multi-series Donut Chart, Title and Figure legend here␣
↪-->
</div>
```

4. Draw the chart in your template by calling *tripalD3.drawChart()*. This is done within a script tag using Drupal behaviours to ensure it is run at the correct point and the data prepared is passed in.

```
<script type="text/javascript">
```

(continues on next page)

(continued from previous page)

```
  Drupal.behaviors.tripalD3demoSimpleDonut = {
    attach: function (context, settings) {

      // Pull the data out of the javascript settings.
      var data = Drupal.settings.demo.stockTypeDonutData;

      // Draw your chart.
      tripalD3.drawFigure(
        multiDonutData,
        {
          "chartType" : "multidonut",
          "elementId": "tripald3-multidonut",
          "height": 250,
          "width": 650,
          "keyPosition": "right",
          "title": "Comparison of allele calls across 3 FBA-1 markers",
          "legend": "The above chart shows the allele ratios for three seperate␣
↪markers assaying the FBA-1 (fictional but amazing) gene.",
          "key": {"title": "Alleles"},
        }
      );
    }
  };
</script>
```

5. There is no step #5; you're done!

3.4 Custom Colour Schemes

Making your own colour scheme available to this module is as simple as implementing a single hook in your custom module. For example, if your module was named *mymodule*, then you would implement *hook_tripald3_color_schemes()* as follows:

```
/**
 * Implements hook_tripald3_color_schemes().
 */
function mymodule_tripald3_color_schemes() {
  $color_schemes = array();

  // This key should be unique across colour schemes.
  $color_schemes['PrpGr'] = array(
    // This name will show up in the colour scheme configuration
    'name' => 'Purple-Green',
    // These are the colours your colour scheme consists of.
    'colors' => array('#3d4051', '#753fb0', '#8f7abf', '#294090', '#6683c3', '#0C6758
↪', '#7AB318', '#A0C55E', '#9fa7a3'),
    'pick order' => array(
      // This is the order of the above colours when picking for a
      // qualitative chart (e.g. heatmap).
      'quantitative' => array(0, 1, 2, 3, 4, 5, 6, 7, 8),
      // This is the order for a categorical chart (e.g. pie chart)
      'categorical' => array(3, 6, 1, 4, 5, 8, 2, 7, 0)
    ),
  );
```

(continues on next page)

(continued from previous page)

```
  return $color_schemes;
}
```

Then just go to the configuration form (Admin » Tripal » Extension Modules » Tripal D3 Diagrams) and choose your colour scheme to see it used in all your diagrams! If you don't yet have diagrams, see your colour scheme in action by clicking on the demo tab.

3.5 Use Colour Schemes in your JS

If you are making a custom chart, you can pick your colours using this API to make sure it is consistent with all your other diagrams! I also highly recommend adding a figure legend to inform your users and keep things consistent!

```
// The colors variable is a simple array of HEX Codes.

// Retrieve the user set colour scheme...

// Categorical: Maximize contrast for showing different categories.
var colors = tripalD3.getColorScheme("categorical");

// Quantitative: for a gradient.
var colors = tripalD3.getColorScheme("quantitative");
```

3.6 Additional Information

- Examples on how to use each chart in `templates/tripald3_demo_page.tpl.php` and viewed at Admin » Tripal » Extension Modules » Tripal D3 Diagrams » Demo
- In-code documentation for all functions and chart types in *js/tripalD3.*.js*

CHAPTER 4

Tripal D3.js API

These docs provide full details of the javascript API including all options available.

4.1 drawFigure

This function is a common set-up function meant to ensure that figures are consistent, as well as, to facilitate common options. Furthermore, it should make it easier for developers as they only need to remember the name of a single function.

Specifically, this function adds the necessary markup for a figure containing chart svg, figure legend and key. It also ensures that any pre-existing chart, legend, key are removed prior to drawing of the chart. Furthermore, it calls the specific chart drawing functions.

Source Code: `js/tripalD3.js`

def `drawFigure(data, options)`:

Used to draw any diagram. Your main entry pointo to the API.

data A javascript object with the data required to draw the chart. The specifics of this object depend on the chart being drawn.

options A javascript object with any of the following keys:

chartType the type of chart to draw; (REQUIRED) one of pedigree, simplepie, simpledonut, multidonut, simplebar.

elementId The ID of the HTML element the diagram should be attached to.

width The width of the drawing canvas (including key and margins) in pixels.

height The height of the drawing canvas (including key and margins) in pixels.

title the title of the figure diagram.

legend a longer description of the diagram to be used as the figure legend following the title. This should include all relevant scientific information as well as instructions on how to interact with the chart.

margin an object with 'top', 'right','bottom','left' keys. Values are in pixels and all four keys must be set.

chartOptions an object containing options to be passed to the chart. See chart documentation to determine what options are available.

drawKey whether or not to draw the key; default is "true".

keyPosition control the position of the key on your figure; supported options include right (default) or left.

keyWidth the key is fixed width; default width is 250 (pixels).

key an object containing additional options for the key. See "drawKey" function for all available options. Some include:

title the title of the key; default "Legend".

margin an object with 'top', 'right','bottom','left' keys. Values are in pixels and all four keys must be set.

4.1.1 drawKey

This function is called by `drawFigure` where `drawKey: true`. Draws a graphical key on an existing diagram to explain the colours and styles used.

Source Code: `js/tripalD3.js`

def `drawKey(data, options):`

Called by `drawFigure` where `drawKey: true`.

data An array of key items where each item is an object with the following keys:

classes the classes attached to the item represented. These are applied to the colored item of the key (circle, rect, path).

groupClasses additional classes to attach to the grouping element in the key. The type of element is added by default.

label The human-readable label for this key item.

type the type of svg element this key item represents. Supported types include: circle, rect, path.

fillColor the color of the circle/rect.

strokeColor the color of the line.

options A javascript object with any of the following keys:

parentId the ID of the parent element containing the SVG to draw the key on (REQUIRED).

elementId the ID to use for the grouping element containing the key.

width the width of the key in pixels (REQUIRED).

height the height of the key in pixels. The default is calculated based on the number of key elements passed in.

margin the margin to use for the key. an object with 'top', 'right', 'bottom','left' keys. Values are in pixels and all four keys must be set.

4.2 drawSimplePie

Warning: This function should not be used directly. Instead use `drawFigure` where `options.chartType = simplepie`

Source Code: `js/tripalD3.pie.js`

def `drawSimplePie(svg, data, options)`:

svg The canvas to draw the pie chart on.

data An array of objects (one object per category or pie-wedge) with the following keys:

label the human-readable label for this category (pie-wdge).

count the number of items in the category. This is used to determine the size of the wedge and MUST BE AN INTEGER.

options An object containing options for this chart. Supported keys include:

width The width of the drawing canvas (including key and margins) in pixels.

height The height of the drawing canvas (including key and margins) in pixels.

maxRadius the outside radius of the pie chart.

donutWidth the width of the donut (difference between inner and outer radius).

timbitRadius the inside radius of the donut.

labelPadding the number of pixels between the series labels and the right edge of the pie chart.

drawKey whether or not to draw the key; default is "true".

4.3 drawSimpleDonut

Warning: This function should not be used directly. Instead use `drawFigure` where `options.chartType = simplepie`

Source Code: `js/tripalD3.pie.js`

def `drawSimpleDonut(svg, data, options)`

Draw a simple donut pie chart.

svg The canvas to draw the pie chart on.

data An array of objects (one object per category or pie-wedge) with the following keys:

label the human-readable label for this category (pie-wdge).

count the number of items in the category. This is used to determine the size of the wedge and MUST BE AN INTEGER.

options An object containing options for this chart. Supported keys include:

width The width of the drawing canvas (including key and margins) in pixels.

height The height of the drawing canvas (including key and margins) in pixels.

maxRadius the outside radius of the pie chart.

donutWidth the width of the donut (difference between inner and outer radius).

timbitRadius the inside radius of the donut.

labelPadding the number of pixels between the series labels and the right edge of the pie chart.

drawKey whether or not to draw the key; default is "true".

4.4 drawMultiDonut

Warning: This function should not be used directly. Instead use `drawFigure` where `options.chartType = simplepie`

Source Code: `js/tripalD3.pie.js`

def `drawMultiDonut(svg, data, options)`

Draw a multi-series Donut Chart

svg The canvas to draw the chart on.

data An array of objects (one object per series or pie ring) with the following keys:

label the human-readable label for the data series. This will be used to label the ring.

parts an array of objects (one object per category or pie-wedge) with the following keys:

label the human-readable label for this category (pie-wdge).

count the number of items in the category. This is used to determine the size of the wedge and MUST BE AN INTEGER.

options A javascript object providing values to customization options. Supported options include:

width The width of the drawing canvas (including key and margins) in pixels.

height The height of the drawing canvas (including key and margins) in pixels.

maxRadius the maximum radius of the pie chart.

donutWidth the width of each ring.

labelPadding the number of pixels between the series labels and the right edge of the pie chart.

drawKey whether or not to draw the key; default is "true".

4.5 drawSimpleBar

Warning: This function should not be used directly. Instead use `drawFigure` where `options.chartType = simplebar`

def `drawSimpleBar(svg, data, options)`

Draw a simple bar chart.

svg The canvas to draw the pie chart on.

data An array of objects (one object per bar) with the following keys:

label the human-readable label for this bar.

count the number of items in the bar.

options An object containing options for this chart. Supported keys include:

xAxisTitle The title of the X-Axis.

yAxisTitle The title of the Y-Axis.

width The width of the drawing canvas (including key and margins) in pixels.

height The height of the drawing canvas (including key and margins) in pixels.

drawKey whether or not to draw the key; default is "false".

xAxisPadding the number of pixels to pad the left side to provide room for the y-axis labels.

yAxisPadding the number of pixels to pad the bottom to provide room for the x-axis labels.

4.6 ellipsisThrobber

Provides an infinite progress throbber in the form of an Ellipsis (3 dots).

Example Usage:

```
throbber = ellipsisThrobber(svg, {'left':50, 'top':50});
setTimeout(function(){ throbber.remove() }, 3000);
```

Source Code: `js/tripalD3.js`

def `ellipsisThrobber(svg, dimensions):`

Created an infinite progress throbber.

svg The svg canvas to draw the throbber on.

dimensions An object specifying the left and top coordinates for the center of the throbber.

returns The D3.js throbber object

4.7 popover

Add information popovers to any set of elements in a D3 diagram. By default the content of the popover will be the `current.name` of each node.

def `popover(options)`:

options A javascript object with any of the following keys:

diagramId the ID of the element containing the svg diagram to attach the popovers to.

Functionality:

`popover.toggle(d)` Toggle the visibility fo the popover.

`popover.show(d)` Draw the popover for a given node.

`popover.hide(d)` Hide the popover for a given node.

4.8 getColorScheme

This function retrieves the HEX codes for a given colour scheme. It is used throughout this API to ensure consistent colours across diagrams.

def `getColorScheme(type, schemeName)`

Retrieve the colours for a given colour scheme.

type The type of scheme to return false; one of quantitative or categorical (REQUIRED).

schemeName The machine name of the color scheme to return the colors of; Defaults to the scheme chosen by the administrator.

returns An array of HEX codes in the order they should be applied to elements.

4.9 placeWatermark

Warning: Although this function places a watermark over diagrams, it is important to note that the watermark can be omitted with advance knowledge of HTML, CSS and DOM. Please be advised to take extra steps to achieve full restriction of content.‘‘

This function places a watermark overlay to diagrams to indicate proprietary content/visualization in a page.

Example Usage:

```
// Draw.
tripalD3.drawFigure(data, {'chartType' : 'multidonut', 'elementId' : 'multidonut', ...
↪.});

// Watermark. Use default image in ``/css/watermark.png`` as watermark.
tripalD3.placeWatermark();

// OR use alternate image.
tripalD3.placeWatermark({
```

(continues on next page)

(continued from previous page)

```
  'watermark' : 'http://www.mysite.ca/assets/logo-watermark.png',
});
```

Source Code: `js/tripalD3.js`

def `placeWatermark(options)`:

Called after drawFigure() instance.

options A javascript object with the following key:

watermark by default this function will use the image located in `/css/watermark.png` as the preselected watermark. This can be

used when site requires a unified watermark across all visualizations (site-wide), simply by replacing the said image with the desired logotype (be sure to use the same filename). If the site requires a different image for one diagram in a page to another diagram in another page, this option can be used to alter the default watermark by supplying a path to an image (see example).

CHAPTER 5

Contributing

We're excited to work with you! Post in the issues queue with any questions, feature requests, or proposals.

5.1 General Goals & Tips

- Highlight your functionality in the demo page (`admin/tripal/extension/tripald3/demo`) without removing any of the charts already there. This makes it easy for me to review and gives you a place to test your chart.
- Ensure your functionality doesn't break existing charts by checking the demo page and confirming all the charts are drawn correctly.
- You can use the test page (`admin/tripal/extension/tripald3/test`) to ensure that data is validated correctly or to check edge cases. This is a great place to demonstrate any bugs you are fixing!
- **For new chart types,**
  - Any new chart types should be called through drawFigure().
  - Colours should be chosen using the Color Scheme API (i.e getColorScheme()) so that all diagrams look consistent.
  - Use popovers included with this API for information provided on hover.