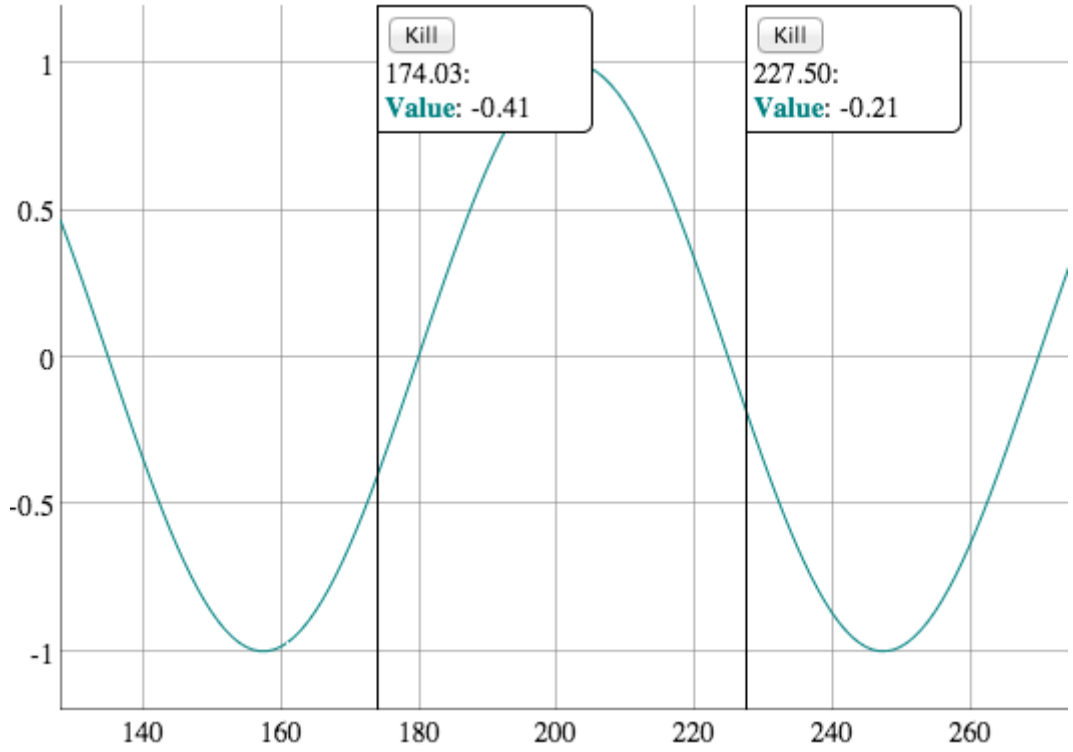
Hairlines

The Hairlines plugin in action:



To use hairlines with a chart, instantiate a Hairlines plugin and pass it in to the constructor:

```
hairlines = new Dygraph.Plugins.Hairlines(options);
dygraph = new Dygraph(div, data, {
    // (other options)
    plugins: [ hairlines ]
});
```

You'll use the `hairlines` object to access information about the hairlines on the chart. You should create one hairline object per Dygraph object.

When you pass the Hairlines plugin to the dygraphs constructor, it enables a few behaviors:

1. Clicking the chart creates a new hairline (double-click still zooms out)
2. Dragging a hairline moves it around
3. Hairlines update on changes in chart data or size (i.e. if the window resizes).

UI

The hairlines always have a "stick" coming down which covers the entire chart vertically. They have a "bubble" on the top which can be completely customized. When a hairline is created, the plugin will look for an element with id=hairline-template and clone it. It should be something like this:

```
<div id="hairline-template" class="hairline-info" style="display:none">
  <button class='hairline-kill-button'>Kill</button>
  <div class='hairline-legend'></div>
</div>
```

Note that it's invisible (so it can be used as a template without affecting the page's display). Note also the two specially-classed elements inside of it:

1. `.hairline-kill-button` Anything in a hairline with this class will, when clicked, remove the hairline. "clicked" here means a jQuery click event. It can be a button, link, etc. Anything that can trigger a 'click' event.
2. `.hairline-legend` Anything in a hairline with this class will be filled in with information about the closest points to the hairline. The UI is identical to the standard dygraphs legend.

You can use CSS to style the hairline. The "bubble" consists entirely your own DOM elements, so you can set classes or styles on it as you wish. The "stick" has the `.dygraph-hairline` class applied to it. You can use the "border-right" style to adjust its appearance. See tests/hairlines.html for details.

If the hairline is selected, both the `.dygraph-hairline` and `#hairline-template` elements will get a "selected" class added to them.l

API

The hairlines object provides a read/write API with event listeners.

hairlines.get()
This method returns an array of hairlines which are currently on the chart. Each has an "xval" property which determines the x-coordinate at which it is displayed.. They also have "selected" and "interpolated" boolean fields.

hairlines.set(lines)
Pass in an array of objects with "xval" properties to update the set of hairlines. This will add new hairlines or destroy existing hairlines as necessary. Calling this method will trigger a "hairlinesChanged" event (see below). You may also adjust the "selected" and "interpolated" fields of each hairline.

Calling `hairlines.set(hairlines.get())` is a no-op, aside from triggering this event.

Calling `hairlines.set([])` will remove all hairlines from the chart.

Event listeners can be attached using jQuery's event system, e.g.

```
$(hairlines).on('hairlinesChanged', function(e, param) {
  // this = hairlines object
  // param = object with additional information about the event
});
```

The events which you can register are:

`hairlinesChanged()`
This is a catch-all event. If anything about the hairlines changes, it will be triggered. It's useful for tracking state changes if you want to persist the hairlines.

`hairlineDeleted({xval})`
A hairline was deleted at the given position.

`hairlineCreated({xval})`
A hairline was created at the given position.

`hairlineMoved({oldXVal, newXVal})`
A hairline was moved from the old position to the new one. This event will fire continuously as the hairline is dragged.

The events are fired after the changes have taken place. Calling this.get() in an event listener will reflect the changes you're being informed of.

Options

The Hairlines class takes one optional constructor argument, a dictionary of options.

The possible options are:

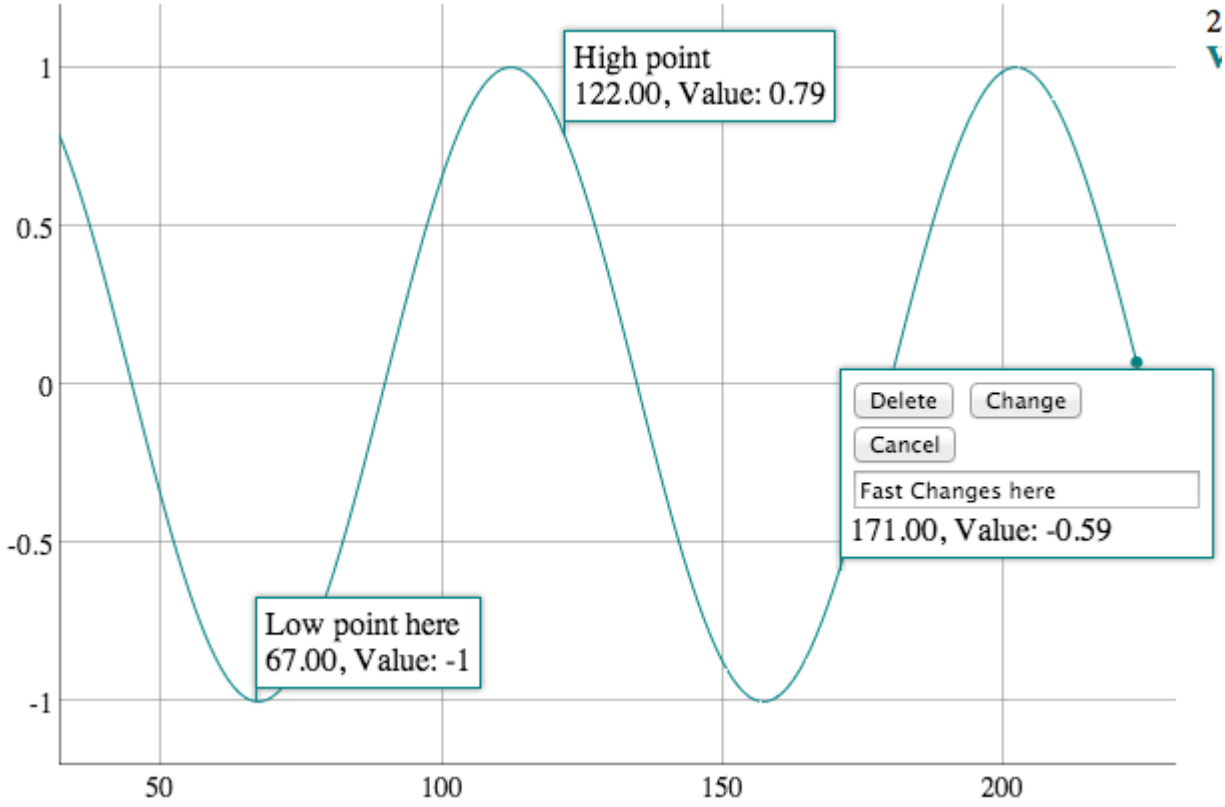
`divFiller: function(div, { closestRow, points, hairline, dygraph })`
The default hairline behavior is to fill the "info div" (i.e. the one at the top of the hairline) with something that looks like the standard dygraphs legend. If you wish to customize this behavior, you may do so by specifying this callback.

`closestRow` is a row number that can be passed to `dygraph.getValue()`. This is only set when the hairline is in "closest" mode (i.e. hairline.interpolated == false).
`points` is an array, identical to what would be returned by `dygraph.getSelection()`. In "interpolated" mode, each point may be synthesized (i.e. it does not occur in the data set). The point objects will have `prevRow` and `nextRow` fields. These indicate the previous/next non-null rows for that series. Either may be null (but not both) if the x-value is off the edge of the series. If the hairline is on a point, then they will be equal.
`hairline` is a Hairline object, as returned by `hairlines.get()`.
`dygraph` is the dygraph object.

This function should fill out the div however it wishes. To get the default behavior, you can call `Dygraph.Plugins.Legend.generateLegendHTML`.

SuperAnnotations

The SuperAnnotations plugin in action:



The SuperAnnotations plugin operates independently from the dygraphs' built-in annotations feature.

It offers a few advantages over the built-in annotations. SuperAnnotations can...

- … be created and edited in-place by users
- … be dragged to different points
- … contain arbitrary metadata
- … use custom UIs

SuperAnnotations exposes an API very similar to the Hairlines API.

```
annotations = new Dygraph.Plugins.SuperAnnotations({
  // this is optional -- annotations have a 'text' property by default.
  defaultAnnotationProperties: {
    'text': 'Annotation Description',
    // other properties annotations should have by default
  }
});
g = new Dygraph(div, data, {
  // (other options)
  plugins: [ annotations ]
});
```

This adds a few capabilities to the chart:

1. Clicking on a point will create a new, editable annotation at that point.
2. Dragging the annotation will attach it to a different point.
3. The annotation will remain attached to that point across redraws and data updates.

Each annotation is an object:

```
{
  series: ‘name of the series to which this annotation is attached’,
  xval: x-value at which it’s attached (value or millis since epoch)
  //  … whatever other properties you wish to set …
}
```

By default, annotations also have a ‘text’ property, but you’re not bound to this.

To make SuperAnnotations work correctly, you’ll need to create two HTML templates: one for a normal annotation and one for an editable annotation. Here’s an example:

```
<!-- Normal Annotation -->
<div id="annotation-template" style="display:none">
  <div>{{text}}</div>
  <div>{{x}}, {{series}}: {{y}}</div>
</div>
```

The template must have an ID of “annotation-template” (*it may change to using a constructor parameter*) and should be hidden by default (as any template is).

When an annotation is created, the SuperAnnotations plugin will loop through all (key, value) pairs for the annotation object and replace {{key}} with the appropriate value. {{x}}, {{y}} and {{series}} are always available. The other keys depend on what you put in the annotations objects. You could add an ‘icon’ field, for example, or ‘alt_text’ or ‘href’.

The editable template works similarly, but with a few extra wrinkles:

```
<!-- Editable Annotation -->
<div id="annotation-editable-template" style="display:none">
  <button class='annotation-kill-button'>Delete</button>
  <button class='annotation-update'>Change</button>
  <button class='annotation-cancel'>Cancel</button><br/>
  <input dg-ann-field='text' type='text' size=30 value='{{text}}' />
  <div>{{x}}, {{series}}: {{y}}</div>
</div>
```

Again, id=”annotation-editable-template” is special. The classes of the buttons are special as well — SuperAnnotations will attach the appropriate behavior to clicks on each. Replacement of {{key}} strings works as before.

The “dg-ann-field” attribute is the other wrinkle. This tells SuperAnnotations to pull the value out of this form element and put it in the “text” property of the annotation object being edited.

If you want to commit changes to an annotation by some other means (e.g. pressing <enter>), you can synthesize a click on the .annotation-update button using jQuery. You can even hide the button if you don’t want to see it.

The events & API work similarly to the Hairlines plugin.

TODO: describe the API & events in more detail

API
.get() -- Returns the list of annotation objects
.set(annotations) -- Sets the annotation for the chart

Events
annotationsChanged
annotationCreated
annotationDeleted
annotationMoved
beganEditAnnotation
annotationEdited
cancelEditAnnotation