sapphire Documentation

Release 0.1.0

Praekelt Foundation

November 03, 2014

Contents

1 api **3**
- 1.1 sapphire.widgets.last . . . 3
- 1.2 sapphire.widgets.lines . . . 6
- 1.3 sapphire.widgets.bars . . . 12
- 1.4 sapphire.widgets.pie . . . 16
- 1.5 sapphire.widgets.widget . . . 19
- 1.6 sapphire.view . . . 20

2 configuration **21**
- 2.1 accessors . . . 21

A js library for drawing dashboards.

CHAPTER 1

api

1.1 sapphire.widgets.last

A widget displaying the last `y` value in a series of datapoints, a sparkline of the values, and a summary of the difference between the last value and the value preceding it.

`sapphire.widgets.`**`last`**`()`
Creates a new last widget.

`last`(*el*)
Draws the widget by applying it to the given selection. `el` can be a d3 selection, or any argument accepted by d3.select.

```
var last = sapphire.widgets.last();

d3.select('#last')
  .datum({
    title: 'A last widget',
    values: [{
      x: 123,
      y: 345
    }, {
      x: 567,
      y: 789
    }]
  })
  .call(last);
```

`last.`**`title`**([*accessor*])
Property for the *accessor* to use to access the widget's title from the bound datum. Defaults to `function(d) { return d.title; }`.

```
var last = sapphire.widgets.last()
  .title(function(d) { return d.heading; });

d3.select('#last')
  .datum({
    ...
    heading: 'A last widget',
    ...
  })
  .call(last);
```

`last.`**`values`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the widget's sets of `x` and `y` values or datapoints. Defaults to `function(d) { return d.values; }`.

```
var last = sapphire.widgets.last()
  .values(function(d) { return d.datapoints; });

d3.select('#last')
  .datum({
    ...
    datapoints: [{
      x: 123,
      y: 345
    }, {
      x: 567,
      y: 789
    }]
    ...
  })
  .call(last);
```

`last.`**`x`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the `x` value from each datum in the array corresponding to `last.values()`. Defaults to `function(d) { return d.x; }`.

```
var last = sapphire.widgets.last()
  .x(function(d) { return d.time; });

d3.select('#last')
  .datum({
    ...
    values: [{
      time: 123,
      y: 345
    }, {
      time: 567,
      y: 789
    }]
    ...
  })
  .call(last);
```

`last.`**`y`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the `y` value from each datum in the array corresponding to `last.values()`. Defaults to `function(d) { return d.y; }`.

```
var last = sapphire.widgets.last()
  .y(function(d) { return d.value; });

d3.select('#last')
  .datum({
    ...
    values: [{
      x: 123,
      value: 345
    }, {
      x: 567,
      value: 789
    }]
```

```
    ...
  })
  .call(last);
```

`last.`**`yFormat`**`([`*fn*`])`
Property for the formatting function to use when displaying the last `y` value. Defaults to `d3.format(',2s')`.

```
var last = sapphire.widgets.last()
  .yFormat(d3.format('.2s'));
```

`last.`**`diffFormat`**`([`*fn*`])`
Property for the formatting function to use when displaying the difference between the last `y` value and the `y` value preceding it. Defaults to `d3.format('+,2s')`.

```
var last = sapphire.widgets.last()
  .diffFormat(d3.format('.2s'));
```

`last.`**`xFormat`**`([`*fn*`])`
Property for the formatting function to use when displaying the last `x` value and the `x` value preceding it. Defaults to `d3.time.format('%-d %b %-H:%M')`.

```
var last = sapphire.widgets.last()
  .xFormat(d3.time.format('%-d %b %-H:%M'));
```

`last.`**`none`**`([`*v*`])`
Property for the value to display as the last value when `last.values()` returns an empty array. Defaults to `0`.

```
var last = sapphire.widgets.last()
  .none(0);
```

`last.`**`width`**`([`*v*`])`
Property for the *accessor* to use to access the widget's width. Used if the widget is standalone. Defaults to `400`.

```
var last = sapphire.widgets.last()
  .width(400);
```

`last.`**`sparklineLimit`**`([`*v*`])`
Property for the minimum number of values or datapoints needed for the sparkline to be drawn. Defaults to `15`.

Note that the given value will be floored at `2`.

```
var last = sapphire.widgets.last()
  .sparklineLimit(3);
```

`last.`**`summaryLimit`**`([`*v*`])`
Property for the minimum number of values or datapoints needed for the summary to be drawn. Defaults to `2`.

Note that the given value will be floored at `2`.

```
var last = sapphire.widgets.last()
  .summaryLimit(3);
```

`last.`**`explicitComponents`**`([`*v*`])`
Property for setting whether the widget should expect its components to be layed out explictly or not.

If set to `false`, the widget will append the components automatically.

If set to `true`, the widget will look for the relevant element's component child elements to decide where to draw each.

Defaults to `false`.

```
<div id="foo">
  <div data-widget-component="title"></div>
  <div data-widget-component="last-value"></div>
  <div data-widget-component="sparkline"></div>
  <div data-widget-component="summary"></div>
 </div>

var last = sapphire.widgets.last()
  .explicitComponents(true);

d3.select("#foo")
  .datum({...})
  .call(last);
```

The last widget's components are:

- `'title'`: title of the widget
- `'last-value'`: text showing the last given y value
- `'sparkline'`: the widget's sparkline summarising the changes in values
- `'summary'`: textual summary of the most recent change in value

1.2 sapphire.widgets.lines

A widget displaying a set of metrics in a line chart, accompanied by a table displaying each metric's title, colour and last `y` value.

`sapphire.widgets.`**`lines`**`()`
Creates a new lines widget.

`lines`(*el*)
Draws the widget by applying it to the given selection. `el` can be a d3 selection, or any argument accepted by d3.select.

```
var lines = sapphire.widgets.lines();

d3.select('#lines')
  .datum({
    key: 'a',
    title: 'Total Foo and Bar',
    metrics: [{
      key: 'foo',
      title: 'Total Foo',
      values: [{
        x: 1405013457677,
        y: 1000000
      }, {
        x: 1405013458677,
        y: 9000000
      }]
    }, {
      key: 'bar',
      title: 'Total Bar',
      values: [{
        x: 1405013457677,
        y: 8000000
```

```
        }, {
          x: 1405013458677,
          y: 3000000
        }]
      }],
    })
    .call(lines);
```

`lines.`**`title`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the widget's title from the bound datum. Defaults to `function(d) { return d.title; }`.

```
var lines = sapphire.widgets.lines()
  .title(function(d) { return d.heading; });

d3.select('#lines')
  .datum({
    heading: 'A lines widget',
    ...
  })
  .call(lines);
```

`lines.`**`metrics`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the widget's array of metrics from the bound datum. Defaults to `function(d) { return d.metrics; }`.

```
var lines = sapphire.widgets.lines()
  .metrics(function(d) { return d.sets; });

d3.select('#lines')
  .datum({
    ...
    sets: [{
     ...
      values: [{
        x: 1405013457677,
        y: 1000000
      }, {
        x: 1405013458677,
        y: 9000000
      }],
      ...
    }, {
      ...
      values: [{
        x: 1405013457677,
        y: 8000000
      }, {
        x: 1405013458677,
        y: 3000000
      }],
      ...
    }]
  })
  .call(lines);
```

`lines.`**`key`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the key of each metric in the array returned by `lines.metrics()`. Defaults to `function(d, i) { return i; }`.

```
var lines = sapphire.widgets.lines()
  .key(function(d) { return d.name; });

d3.select('#lines')
  .datum({
    ...
    metrics: [{
      ...
      name: 'Foo',
      ...
    }, {
      ...
      name: 'Bar',
      ...
    }]
  })
  .call(lines);
```

`lines.`**`metricTitle`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the title of each metric in the array returned by `lines.metrics()`. Defaults to `function(d) { return d.title; }`.

```
var lines = sapphire.widgets.lines()
  .metricTitle(function(d) { return d.name; });

d3.select('#lines')
  .datum({
    ...
    metrics: [{
      ...
      name: 'Foo',
      ...
    }, {
      ...
      name: 'Bar',
      ...
    }]
  })
  .call(lines);
```

`lines.`**`values`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the sets of `x` and `y` values or datapoints from each item in the array returned by `lines.metrics()`. Defaults to `function(d) { return d.values; }`.

```
var lines = sapphire.widgets.lines()
  .values(function(d) { return d.datapoints; });

d3.select('#lines')
  .datum({
    ...
    metrics: [{
      ...
      datapoints: [{
        x: 1405013457677,
        y: 1000000
      }, {
        x: 1405013458677,
        y: 9000000
      }],
```

```
        ...
      }, {
        ...
        datapoints: [{
          x: 1405013457677,
          y: 8000000
        }, {
          x: 1405013458677,
          y: 3000000
        }],
        ...
      }]
    })
    .call(lines);
```

`lines.`**`x`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the `x` value from each datum in the array returned by `lines.values()`. Defaults to `function() { return d.x; }`.

```
var lines = sapphire.widgets.lines()
  .x(function(d) { return d.time; });

d3.select('#lines')
  .datum({
    ...
    metrics: [{
      ...
      datapoints: [{
        time: 1405013457677,
        y: 1000000
      }, {
        time: 1405013458677,
        y: 9000000
      }],
      ...
    }, {
      ...
      datapoints: [{
        time: 1405013457677,
        y: 8000000
      }, {
        time: 1405013458677,
        y: 3000000
      }],
      ...
    }]
    ...
  })
  .call(lines);
```

`lines.`**`y`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the `y` value from each datum in the array corresponding to `lines.values()`. Defaults to `function() { return d.y; }`.

```
var lines = sapphire.widgets.lines()
  .y(function(d) { return d.value; });

d3.select('#lines')
  .datum({
```

```
    ...
    metrics: [{
      ...
      values: [{
        x 1405013457677,
        value: 1000000
      }, {
        x 1405013458677,
        value: 9000000
      }],
      ...
    }, {
      ...
      values: [{
        x 1405013457677,
        value: 8000000
      }, {
        x 1405013458677,
        value: 3000000
      }],
      ...
    }]
    ...
  })
  .call(lines);
```

`lines.`**`yFormat`**`(`[*fn*]`)`

Property for the formatting function to use when displaying the last `y` value. Defaults to `d3.format(',2s')`.

```
var lines = sapphire.widgets.lines()
  .yFormat(d3.format('.2s'));
```

`lines.`**`xTickFormat`**`(`[*fn*]`)`

Property for the formatting function to use when displaying the tick values on the line chart's `x` axis. Defaults to `null`.

`sapphire.widgets.lines()` uses d3.time.scale to generate its time scale, so when `lines.xFormat()` is `null`, the built-in d3 tick formatter is used.

```
var lines = sapphire.widgets.lines()
  .xFormat(d3.time.format('%Y-%m-%d'));
```

`lines.`**`xTicks`**`(`[*v*]`)`

Property for the number of ticks to use for the x axis of the chart. This is given directly to d3.time.scale. Defaults to `8`.

```
var lines = sapphire.widgets.lines()
  .xTicks(10);
```

`lines.`**`yTickFormat`**`(`[*fn*]`)`

Property for the formatting function to use when displaying the tick values on the line chart's `y` axis. Defaults to `d3.format('.2s')`.

```
var lines = sapphire.widgets.lines()
  .yFormat(d3.format('s'));
```

`lines.`**`yTicks`**`(`[*v*]`)`

Property for the number of ticks to use for the y axis of the chart. This is given directly to d3.time.scale. Defaults to `5`.

```
var lines = sapphire.widgets.lines()
  .yTicks(10);
```

`lines.`**`yMin`**`(`[*v*]`)`

Property for the chart's minimum y axis value. If a number is given, its value will be used as the chart's minimum value. If a function is given, the function will be passed an array of the y values to display and should return the number to use as the minimum. Defaults to `d3.min`.

```
var lines = sapphire.widgets.lines()
  .yMin(function(values) {
    return d3.min([9000].concat(values));
  });
```

`lines.`**`yMax`**`(`[*v*]`)`

Property for the chart's maximum y axis value. If a number is given, its value will be used as the chart's maximum value. If a function is given, the function will be passed an array of the y values to display and should return the number to use as the maximum. Defaults to `d3.max`.

```
var lines = sapphire.widgets.lines()
  .yMax(function(values) {
    return d3.max([9000].concat(values));
  });
```

`lines.`**`colors`**`(`[*fn*]`)`

Property for the colour function to use to calculate each metric's colour from the values returned by `lines.keys()`. Defaults to `d3.scale.category10()`.

```
var lines = sapphire.widgets.lines()
  .colors(d3.scale.category10());
```

`lines.`**`none`**`(`[*v*]`)`

Property for the value to display as the last value when `lines.values()` returns an empty array. Defaults to `0`.

```
var lines = sapphire.widgets.lines()
  .none(0);
```

`lines.`**`width`**`(`[*v*]`)`

Property for the *accessor* to use to access the widget's width. Used if the widget is standalone. Defaults to `400`.

```
var lines = sapphire.widgets.lines()
  .width(400);
```

`lines.`**`explicitComponents`**`(`[*v*]`)`

Property for setting whether the widget should expect its components to be layed out explictly or not.

If set to `false`, the widget will append the components automatically.

If set to `true`, the widget will look for the relevant element's component child elements to decide where to draw each.

Defaults to `false`.

```
<div id="foo">
  <div data-widget-component="title"></div>
  <div data-widget-component="chart"></div>
  <div data-widget-component="legend"></div>
 </div>
```

```
var lines = sapphire.widgets.lines()
  .explicitComponents(true);

d3.select("#foo")
  .datum({...})
  .call(lines);
```

The lines widget's components are:

- `'title'`: title of the widget
- `'chart'`: the actual line chart
- `'legend'`: table showing the color, title and values of each metric

1.3 sapphire.widgets.bars

A widget for displaying time-based data on consecutive bars on a chart, where each bar corresponds to a time interval.

`sapphire.widgets.`**`bars`**`()`
Creates a new bars widget.

`bars`(*el*)
Draws the widget by applying it to the given selection. `el` can be a d3 selection, or any argument accepted by d3.select.

```
var bars = sapphire.widgets.bars();

d3.select('#bars')
  .datum({
    title: 'Total Foo',
    values: [{
      x: 1405013457677,
      y: 1000000
    }, {
      x: 1405013458677,
      y: 9000000
    }]
  })
  .call(bars);
```

`bars.`**`title`**([*accessor*])
Property for the *accessor* to use to access the widget's title from the bound datum. Defaults to `function(d) { return d.title; }`.

```
var bars = sapphire.widgets.bars()
  .title(function(d) { return d.heading; });

d3.select('#bars')
  .datum({
    heading: 'A bars widget',
    ...
  })
  .call(bars);
```

`bars.`**`values`**([*accessor*])
Property for the *accessor* to use to access the sets of `x` and `y` values or datapoints from the bound datum.

Defaults to `function(d) { return d.values; }`.

```
var bars = sapphire.widgets.bars()
  .values(function(d) { return d.datapoints; });

d3.select('#bars')
  .datum({
    ...
    datapoints: [{
      x: 1405013457677,
      y: 1000000
    }, {
      x: 1405013458677,
      y: 9000000
    }]
  })
  .call(bars);
```

`bars.`**`x`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the `x` value from each datum in the array returned by `bars.values()`. Defaults to `function() { return d.x; }`.

```
var bars = sapphire.widgets.bars()
  .x(function(d) { return d.time; });

d3.select('#bars')
  .datum({
    ...
    metrics: [{
      ...
      datapoints: [{
        time: 1405013457677,
        y: 1000000
      }, {
        time: 1405013458677,
        y: 9000000
      }],
      ...
    }, {
      ...
      datapoints: [{
        time: 1405013457677,
        y: 8000000
      }, {
        time: 1405013458677,
        y: 3000000
      }],
      ...
    }]
    ...
  })
  .call(bars);
```

`bars.`**`y`**`(`[*accessor*]`)`

Property for the *accessor* to use to access the `y` value from each datum in the array corresponding to `bars.values()`. Defaults to `function() { return d.y; }`.

```
var bars = sapphire.widgets.bars()
  .y(function(d) { return d.value; });
```

```
d3.select('#bars')
  .datum({
    ...
    metrics: [{
      ...
      values: [{
        x 1405013457677,
        value: 1000000
      }, {
        x 1405013458677,
        value: 9000000
      }],
      ...
    }, {
      ...
      values: [{
        x 1405013457677,
        value: 8000000
      }, {
        x 1405013458677,
        value: 3000000
      }],
      ...
    }]
    ...
  })
  .call(bars);
```

bars.**xTickFormat**([*fn*])

Property for the formatting function to use when displaying the tick values on the line chart's x axis. Defaults to null.

sapphire.widgets.bars() uses d3.time.scale to generate its time scale, so when bars.xFormat() is null, the built-in d3 tick formatter is used.

```
var bars = sapphire.widgets.bars()
  .xFormat(d3.time.format('%Y-%m-%d'));
```

bars.**xTicks**([*v*])

Property for the number of ticks to use for the x axis of the chart. This is given directly to d3.time.scale. Defaults to 8.

```
var bars = sapphire.widgets.bars()
  .xTicks(10);
```

bars.**yTickFormat**([*fn*])

Property for the formatting function to use when displaying the tick values on the line chart's y axis. Defaults to d3.format('.2s').

```
var bars = sapphire.widgets.bars()
  .yFormat(d3.format('s'));
```

bars.**yTicks**([*v*])

Property for the number of ticks to use for the y axis of the chart. This is given directly to d3.time.scale. Defaults to 5.

```
var bars = sapphire.widgets.bars()
  .yTicks(10);
```

bars.**yMax**([*v*])
Property for the chart's maximum y axis value. If a number is given, its value will be used as the chart's maximum value. If a function is given, the function will be passed an array of the y values to display and should return the number to use as the maximum. Defaults to `d3.max`.

```
var bars = sapphire.widgets.bars()
  .yMax(function(values) {
    return d3.max([9000].concat(values));
  });
```

bars.**colors**([*fn*])
Property for the colour function to use to calculate the colour used for the chart's bars, where the result of `bars.title()` is used as input to the function. Defaults to `d3.scale.category10()`.

```
var bars = sapphire.widgets.bars()
  .colors(d3.scale.category10());
```

bars.**width**([*v*])
Property for the *accessor* to use to access the widget's width. Used if the widget is standalone. Defaults to `400`.

```
var bars = sapphire.widgets.bars()
  .width(400);
```

bars.**height**([*v*])
Property for the *accessor* to use to access the widget's height. Used if the widget is standalone. Defaults to `200`.

```
var bars = sapphire.widgets.bars()
  .height(200);
```

bars.**explicitComponents**([*v*])
Property for setting whether the widget should expect its components to be layed out explictly or not.

If set to `false`, the widget will append the components automatically.

If set to `true`, the widget will look for the relevant element's component child elements to decide where to draw each.

Defaults to `false`.

```
<div id="foo">
  <div data-widget-component="title"></div>
  <div data-widget-component="chart"></div>
 </div>

var bars = sapphire.widgets.bars()
  .explicitComponents(true);

d3.select("#foo")
  .datum({...})
  .call(bars);
```

The bars widget's components are:

- `'title'`: title of the widget
- `'chart'`: the actual bar chart

1.4 sapphire.widgets.pie

A widget displaying a set of metrics on a pie chart, along with a table displaying each metric's title, colour, value and percentage.

`sapphire.widgets.`**`pie`**`()`
Creates a new pie widget.

`pie``(`*el*`)`
Draws the widget by applying it to the given selection. `el` can be a d3 selection, or any argument accepted by d3.select.

```
var pie = sapphire.widgets.pie();

d3.select('#pie')
.datum({
  title: 'Corge, Grault and Garply',
  metrics: [{
    key: 'Corge',
    title: 'Corge',
    value: 89251
  }, {
    key: 'Grault',
    title: 'Grault',
    value: 21479
  }, {
    key: 'Garply',
    title: 'Garply',
    value: 76432
  }]
})
.call(pie);
```

`pie.`**`title`**`(`[*accessor*]`)`
Property for the *accessor* to use to access the widget's title from the bound datum. Defaults to `function(d) { return d.title; }`.

```
var pie = sapphire.widgets.pie()
  .title(function(d) { return d.heading; });

d3.select('#pie')
  .datum({
    heading: 'A pie widget',
    ...
  })
  .call(pie);
```

`pie.`**`metrics`**`(`[*accessor*]`)`
Property for the *accessor* to use to access the widget's array of metrics from the bound datum. Defaults to `function(d) { return d.metrics; }`.

```
var pie = sapphire.widgets.pie()
  .metrics(function(d) { return d.sets; });

d3.select('#pie')
  .datum({
    ...
    sets: [{
     ...
```

```
        value: 1000000
      }, {
        ...
        value: 3000000
        ...
      }]
    })
    .call(pie);
```

pie.**key**([*accessor*])

Property for the *accessor* to use to access the key of each metric in the array returned by `pie.metrics()`. Defaults to `function(d, i) { return i; }`.

```
var pie = sapphire.widgets.pie()
  .key(function(d) { return d.name; });

d3.select('#pie')
  .datum({
    ...
    metrics: [{
      ...
      name: 'Foo',
      ...
    }, {
      ...
      name: 'Bar',
      ...
    }]
  })
  .call(pie);
```

pie.**metricTitle**([*accessor*])

Property for the *accessor* to use to access the title of each metric in the array returned by `pie.metrics()`. Defaults to `function(d) { return d.title; }`.

```
var pie = sapphire.widgets.pie()
  .metricTitle(function(d) { return d.name; });

d3.select('#pie')
  .datum({
    ...
    metrics: [{
      ...
      name: 'Foo',
      ...
    }, {
      ...
      name: 'Bar',
      ...
    }]
  })
  .call(pie);
```

pie.**value**([*accessor*])

Property for the *accessor* to use to access the values to display on the pie chart from each item in the array returned by `pie.metrics()`. Defaults to `function(d) { return d.value; }`.

```
var pie = sapphire.widgets.pie()
  .value(function(d) { return d.values[0].y; });

d3.select('#pie')
  .datum({
    ...,
    metrics: [{
      ...,
      values: [{
        ...,
        y: 1000000
      }]
    }, {
      ...,
      values: [{
        ...,
        y: 8000000
      }]
    }],
  })
  .call(pie);
```

pie.**valueFormat**([*fn*])

Property for the formatting function to use when displaying the metric values in the widget's table. Defaults to `d3.format(',2s')`.

```
var pie = sapphire.widgets.pie()
  .valueFormat(d3.format('s'));
```

pie.**percentFormat**([*fn*])

Property for the formatting function to use when displaying the metric percentages in the widget's table. Defaults to `d3.format('.0%')`.

```
var pie = sapphire.widgets.pie()
  .percentFormat(d3.format('.1%'));
```

pie.**colors**([*fn*])

Property for the colour function to use to calculate each metric's colour from the values returned by `pie.keys()`. Defaults to

`d3.scale.category10()`.

```
var pie = sapphire.widgets.pie()
  .colors(d3.scale.category10());
```

pie.**width**([*v*])

Property for the *accessor* to use to access the widget's width. Used if the widget is standalone. Defaults to `400`.

```
var pie = sapphire.widgets.pie()
  .width(400);
```

pie.**innerRadius**([*v*])

Property for setting the pie chart's inner radius. If a function is given, the function is invoked with the pie chart's outer radius. Defaults to `0`.

```
var pie = sapphire.widgets.pie()
  .innerRadius(0);
```

pie.**explicitComponents**([*v*])

Property for setting whether the widget should expect its components to be layed out explictly or not.

If set to `false`, the widget will append the components automatically.

If set to `true`, the widget will look for the relevant element's component child elements to decide where to draw each.

Defaults to `false`.

```
<div id="foo">
  <div data-widget-component="title"></div>
  <div data-widget-component="chart"></div>
  <div data-widget-component="legend"></div>
 </div>

var pie = sapphire.widgets.pie()
  .explicitComponents(true);

d3.select("#foo")
  .datum({...})
  .call(pie);
```

The pie widget's components are:

- `'title'`: title of the widget
- `'chart'`: the actual pie chart
- `'legend'`: table showing the color, title and values of each metric

1.5 sapphire.widgets.widget

The base widget type to extend to define a widget type. `sapphire.widgets.widget()` extends `sapphire.view()`, so methods such as *view.extend* and `view.draw()` are also available on widget types.

`sapphire.widgets.`**`widget`**`()`
Creates a new widget.

`widget``(`*el*`)`
Draws the widget by applying it to the given selection.

```
var widget = widget.widgets.widget();

d3.select('#widget')
  .datum({title: 'A widget'})
  .call(widget);
```

`widget.`**`width`**`(`[*v*]`)`
Property for the *accessor* to use to access the widget's width. Used if the widget is standalone. Defaults to `100`.

```
var widget = sapphire.widget()
  .width(100);
```

`widget.`**`height`**`(`[*v*]`)`
Property for the *accessor* the widget's height. Used if the widget is standalone. Defaults to `100`.

Note that widgets may exceed this height, depending on the behaviour of the widget type. For example, `sapphire.widgets.lines()` has a dynamic height to support the dynamic height of its legend table.

```
var widget = sapphire.widget()
  .height(1);
```

1.6 sapphire.view

A lightweight view component used as a base for sapphire's other components. The component is defined using strain, so strain methods such as `.extend()` and `.prop()` are also available on view components.

`sapphire.`**`view`**`()`
Creates a new view.

`sapphire.view.`**`draw`**`(`*fn*`)`
Defines a new draw method for the view type using the given function. The default drawing function is a no-op, this should be overriden with the drawing instructions specific to a component.

```
var viewtype = sapphire.view.extend()
  .draw(function(el) {
    el.text(function(d) { return d.text; });
  });
```

`sapphire.view.`**`enter`**`(`*fn*`)`
Defines a new enter method for the view type using the given function that will be called on the first `draw()`. The default enter function is a no-op, this can be overriden with the drawing instructions specific to a component, if necessary.

```
var viewtype = sapphire.view.extend()
  .enter(function(el) {
    el.append('div')
      .attr('class', 'foo');
  })
  .draw(function(el) {
    el.select('.foo')
      .text('bar');
  });
```

`view``(`*el*`)`
Draws the view by applying it to the given selection. `el` can be a d3 selection, or any argument accepted by d3.select.

```
var view = sapphire.view.extend()
  .draw(function(el) {
    el.text(function(d) { return d.text; });
  }));
  .new();

d3.select('body')
  .datum({text: 'foo'})
  .call(view);
```

`view.`**`draw`**`(`*el*`)`
Identical to `view()`.

CHAPTER 2

configuration

2.1 accessors

Similar to d3, sapphire allows accessors to be given to specify the values of widget properties.

Accessors be can specified as their actual values:

```
var bars = sapphire.bars()
  .title('A Humble Bar Chart');
```

Or as a function that, given the current datum (`d`), index (`i`) and element (`this`), returns the desired value:

```
var bars = sapphire.bars()
  .title(function(d, i) { return d.title; });
```

Unless otherwise specified, the element used as the `this` context corresponds to the DOM node(s) in the selection that the component was called on:

```
var el = d3.select('#bars');

var bars = sapphire.bars()
  .title(function(d, i) {
    console.log(this === el.node());  // true
    return d.title;
  });

bars(el);
```

B

bars() (built-in function), 12
bars.colors() (built-in function), 15
bars.explicitComponents() (built-in function), 15
bars.height() (built-in function), 15
bars.title() (built-in function), 12
bars.values() (built-in function), 12
bars.width() (built-in function), 15
bars.x() (built-in function), 13
bars.xTickFormat() (built-in function), 14
bars.xTicks() (built-in function), 14
bars.y() (built-in function), 13
bars.yMax() (built-in function), 14
bars.yTickFormat() (built-in function), 14
bars.yTicks() (built-in function), 14

L

last() (built-in function), 3
last.diffFormat() (built-in function), 5
last.explicitComponents() (built-in function), 5
last.none() (built-in function), 5
last.sparklineLimit() (built-in function), 5
last.summaryLimit() (built-in function), 5
last.title() (built-in function), 3
last.values() (built-in function), 3
last.width() (built-in function), 5
last.x() (built-in function), 4
last.xFormat() (built-in function), 5
last.y() (built-in function), 4
last.yFormat() (built-in function), 5
lines() (built-in function), 6
lines.colors() (built-in function), 11
lines.explicitComponents() (built-in function), 11
lines.key() (built-in function), 7
lines.metrics() (built-in function), 7
lines.metricTitle() (built-in function), 8
lines.none() (built-in function), 11
lines.title() (built-in function), 7
lines.values() (built-in function), 8
lines.width() (built-in function), 11
lines.x() (built-in function), 9
lines.xTickFormat() (built-in function), 10
lines.xTicks() (built-in function), 10
lines.y() (built-in function), 9
lines.yFormat() (built-in function), 10
lines.yMax() (built-in function), 11
lines.yMin() (built-in function), 11
lines.yTickFormat() (built-in function), 10
lines.yTicks() (built-in function), 10

P

pie() (built-in function), 16
pie.colors() (built-in function), 18
pie.explicitComponents() (built-in function), 18
pie.innerRadius() (built-in function), 18
pie.key() (built-in function), 17
pie.metrics() (built-in function), 16
pie.metricTitle() (built-in function), 17
pie.percentFormat() (built-in function), 18
pie.title() (built-in function), 16
pie.value() (built-in function), 17
pie.valueFormat() (built-in function), 18
pie.width() (built-in function), 18

S

sapphire.view() (built-in function), 20
sapphire.view.draw() (built-in function), 20
sapphire.view.enter() (built-in function), 20
sapphire.widgets.bars() (built-in function), 12
sapphire.widgets.last() (built-in function), 3
sapphire.widgets.lines() (built-in function), 6
sapphire.widgets.pie() (built-in function), 16
sapphire.widgets.widget() (built-in function), 19

V

view() (built-in function), 20
view.draw() (built-in function), 20

W

widget() (built-in function), 19

widget.height() (built-in function), 19
widget.width() (built-in function), 19