

---

# **SODA**

***Release 1.0***

**Jack Roddy**

**Aug 26, 2022**



# CONTENTS

<b>1</b>	<b>Guide</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Installation and setup . . . . .	2
1.3	Overview . . . . .	3
1.4	Annotations . . . . .	3
1.5	Charts . . . . .	5
1.6	Loading annotation data . . . . .	9
1.7	Rendering . . . . .	17
<b>2</b>	<b>Examples</b>	<b>23</b>
2.1	Simple rectangles . . . . .	24
2.2	Styled rectangles . . . . .	24
2.3	Dynamic text . . . . .	25
2.4	Interactivity . . . . .	26
2.5	Plot annotations . . . . .	27
2.6	Sequence annotations . . . . .	28
<b>3</b>	<b>Api</b>	<b>29</b>
3.1	Classes . . . . .	29
3.2	Interfaces . . . . .	76
3.3	Functions . . . . .	172
3.4	Enumerations . . . . .	185



## 1.1 Introduction

SODA is a lightweight TypeScript/Javascript library for building dynamic and interactive visualizations of biological sequence annotation. Visualizations produced by SODA can be easily integrated with web pages, and it is easy to define interactions between SODA components and other page features.

### 1.1.1 Before you start

SODA is still in the early stages of its life and is currently maintained by one person. If you encounter any bugs, find anything in the library or documentation confusing, or even think there are gaps in the feature set, *please* consider [submitting an issue](#).

SODA adheres to the [semantic versioning](#) guidelines, so any (intentional) breaking changes to the API will be accompanied by a bump in the major version number.

### 1.1.2 Design philosophies

The development of SODA is guided by a handful of design philosophies:

#### **SODA is developed in TypeScript**

Types make code safer, easier to understand, and less painful to maintain.

TypeScript does a fantastic job of adding static typing to JavaScript. If you're not familiar with TypeScript, check out the [TypeScript handbook](#).

Of course, you are still free to use SODA as a JavaScript library, but you'll miss out on a bit of safety.

#### **SODA features use callback functions**

If you've spent time writing JavaScript, it's a safe bet that you're familiar with the concept of a callback function. However, if you've never used callback functions before, it's probably worth taking a quick moment to check out the MDN Web Docs section on [callback functions](#).

Callback functions are used throughout SODA for interactivity and dynamic styling.

#### **SODA is not a visualization tool—it is a library with which visualization tools can be built**

There are countless tools that provide out of the box solutions for visualizing sequence annotation; SODA is not one of those tools. Although there are many common visualization patterns for annotation data, there will always be edge case scenarios with visualization needs that don't quite fit into one of those patterns. For developers who find themselves in one of those scenarios, SODA aims to provide an option that they might find a bit more palatable than turning to a low-level visualization library like D3.

## **SODA makes few assumptions about your data and the way it should be visualized**

SODA never tries to make stylistic decisions for you. Instead, you are in control of deciding how data is visually represented and how that representation changes in response to interactions with the visualization. The only assumption that SODA makes about your data is that it describes annotations along one dimension (e.g. a genome).

## **1.2 Installation and setup**

SODA is implemented in TypeScript, which means it can be used in both TypeScript and JavaScript.

### **1.2.1 SODA as a TypeScript library**

To get the full benefit of TypeScript when using SODA, you'll probably want to use it in an [npm](#) project.

If you have never used npm before, you'll first need to install [Node](#). Depending on your operating system, there may be several ways to do that. Regardless of which platform you are on, you should be able to install it from the [Node homepage](#)

Alternatively, you could install node with a package manager:

**Homebrew:**

```
brew install node
```

**Apt (Ubuntu, Debian):**

```
sudo apt install nodejs
```

After installing node, you can initialize a directory as an npm project:

```
mkdir my-project/  
cd my-project/  
npm init
```

Once you have an npm project, brand new or otherwise, you can install SODA:

```
npm install @sodaviz/soda
```

If you want to, you can instead download the SODA source code from the [GitHub repository](#) and compile it with the TypeScript compiler, [tsc](#).

### **1.2.2 SODA as a JavaScript library**

If you'd rather just use SODA as a JavaScript library, the easiest way is probably to grab the [bundle from skypack](#).

You could also download the source code from the [GitHub repository](#), compile it, and bundle it yourself with something like [webpack](#).

## 1.3 Overview

SODA is a front-end library for visualizing biological sequence annotations in a web page. The SODA API allows you to create SVG viewports in a web page and to configure and manipulate visualizations in those viewports. SODA has no back-end and doesn't create any wrapping GUI components: it's up to you to implement these as you see fit.

Building a simple SODA application requires three pieces:

1. A TypeScript/JavaScript object definition that extends SODA's simple Annotation object and describes the annotation data you want to visualize.
2. One or more SODA Chart objects configured to produce the visualization itself.
3. Top-level code that drives the application by instantiating Annotation objects and delivering them to Chart objects for visualization.

The goal of this guide is to explain the nuances of these systems in detail.

## 1.4 Annotations

Developing with SODA revolves around rendering glyphs using Annotation objects, which are the data structures used by SODA to represent annotation data. Typically, the first thing you'll want to do when building a SODA visualization is to figure out how you're going to load your data into objects that conform to SODA's expectations. This section describes Annotation object structures and their nuances. For details on how Annotation objects are actually used, see the sections on rendering and interactivity.

### 1.4.1 Annotation interfaces

SODA features are designed to use objects that implement the simple interfaces described in this section.

#### Annotation

The *Annotation* interface is the baseline for all Annotation objects in SODA. Objects that implement Annotation can be used to render glyphs that represent intervals in a sequence domain (e.g a gene).

```
interface Annotation {
  id: string;    // <- this should be a unique identifier
  start: number; // <- the start of the interval that the annotation describes
  end: number;   // <- the end of the interval that the annotation describes
}
```

#### PlotAnnotation

Objects that implement PlotAnnotation can be used to render glyphs that represent intervals in a sequence domain for which there are position specific real number values (e.g. GC content).

```
interface PlotAnnotation extends Annotation {
  id: string;
  start: number;
  end: number;
  values: number[]; // <- these are the position specific values
}
```

## SequenceAnnotation

Objects that implement SequenceAnnotation can be used to render glyphs that represent intervals in a sequence domain for which there are position specific character values (e.g. base pairs aligned to a reference sequence).

```
interface SequenceAnnotation extends Annotation {  
  id: string;  
  start: number;  
  end: number;  
  sequence: string; // <- this should contain the character values  
}
```

### 1.4.2 Annotation utilities

SODA provides some utilities that may help you work with Annotation objects:

- *AnnotationGroup* is an object that makes it easy to treat a group of Annotation objects as a single annotation.
- *generateAnnotations* generates some simple, toy DefaultAnnotation objects that can be used for experimentation.
- *aggregateTransitive* creates a list of AnnotationGroups from a supplied list of Annotations and a criterion function that takes a pair of Annotations as arguments. If the criterion function returns true for a pair of Annotations, they are placed in the same group. It is assumed that the criterion can be transitively applied.
- *aggregateIntransitive* creates a list of AnnotationGroups from a supplied list of Annotations and a criterion function that takes a pair of Annotations as arguments. If the criterion function returns true for a pair of Annotations, they are placed in the same group. It is *not* assumed that the criterion can be transitively applied.
- *getAlignmentAnnotations* creates SequenceAnnotations that are suitable for rendering sequences that are aligned to a chromosome.
- *sliceSequenceAnnotation* returns a smaller piece of a SequenceAnnotation.
- *slicePlotAnnotation* returns a smaller piece of a PlotAnnotation.

We're aware that the scope of these utilities may seem a bit lacking. That's because these are the data manipulation utilities that *we* have found useful when developing our own SODA applications. If you think a fundamental utility is missing, please [let us know](#)!

## Established data formats

While SODA is not specifically designed to visualize established annotation data formats, a few utilities are provided to offer light support:

- *parseGff3Records* function can be used to parse GFF3 record strings into *Gff3Annotation* objects.
- *parseBedRecords* function may be used to parse BED record strings into *BedAnnotation* objects.

We'll likely expand light support for more data formats in the future.



## 1.5 Charts

Chart objects are wrappers around SVG viewports in the browser that house SODA visualizations. This section explains both how the viewports are structured and how their wrapper Chart objects work under the hood.

### 1.5.1 Chart anatomy

In the the DOM, Charts exist as a collection of SVG elements inside of a containing div element. By default, the div container is assigned to be 100% of the width of its parent, and its height is dynamically adjusted to properly display the contents of the SVG elements.

We refer to the outermost SVG as the **SVG pad**, and it is set to 100% of the dimensions of the div container. The SVG pad has two child SVG elements, the **viewport** and the **overflow viewport**. By default, both viewport SVG elements are centered in the SVG pad using the Chart's `padSize` property.

The SODA rendering API renders glyphs by placing SVG shapes inside either of the SVG viewports. The viewports are different only in the way that they treat glyphs positioned outside of their bounding box. The **viewport** hides glyphs that bleed into the SVG pad bounding box, while the **overflow viewport** does not. Conceptually, we divide Chart viewports vertically into rows, and, by default, glyphs are sized to fit into these rows. For more details on the rendering, see the [rendering section](#).

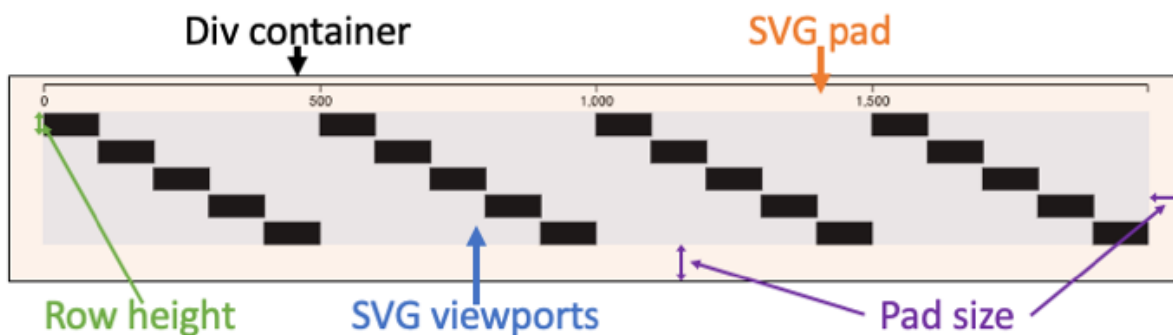


Fig. 1: A Chart with rectangle glyphs in five rows and an axis glyph rendered in the overflow viewport.

### 1.5.2 Chart configuration

Charts are configured and instantiated with a *ChartConfig* object. Excluding *selector*, every property on the *ChartConfig* is optional. The *selector* property is used as a CSS selector to identify the target DOM container that the Chart will be placed in. Typically, the target of the *selector* should be a div that you explicitly create and position to accommodate your Chart viewport.

For example, a minimal Chart configuration may look something like this:

```
let chart = new Chart({ selector: "div#soda-chart" });
```

Running this code will initialize a Chart in a div with the class “soda-chart,” assuming that such a div exists in the DOM. The Chart will be a completely blank slate, which means that you won’t actually see anything rendered in it.

## Dimensions configuration

The majority of properties control the dimensions of the various components of the Chart described above. The dimensions options that are likely to have the most impact on your visualization are *rowHeight* and *padSize*. For a complete list of dimension configuration options, check out the [ChartConfig](#) page in the API documentation.

## Rendering configuration

There are several callback function properties on the `ChartConfig` that, when supplied, override the default behavior of the `Chart.render()` method. The [rendering section](#) describes the rendering processing and the purpose of these callback functions in detail, but we'll note them here for the sake of completeness:

- `updateLayout()`
- `updateRowCount()`
- `updateDimensions()`
- `updateDomain()`
- `draw()`

## Default axes

A default horizontal axis is rendered in the **overflow viewport** if the `chart.addAxis()` function is called. Specifically, it is positioned in the bounding box of the upper section of the **SVG pad**, outside of the bounding box of the **viewport**. The default `chart.draw()` implementation calls the `addAxis()` function.

For example, the following code:

```
let chart = new Chart({
  selector: "div#soda-chart",
});

chart.render({});
```

will produce something like:



Fig. 2: A blank Chart set up to render a default horizontal axis.

Adjusting the *padSize* may cause the axis to be positioned such that it is too low or too high.

If the default horizontal axis doesn't work for your visualization, you can place a horizontal axis wherever you'd like using the [horizontalAxis](#) function.

## Row colors

If the `rowColors` property is configured, the rows in the Chart will be rendered with those colors in a repeating pattern. If the `rowOpacity` property is configured, the value will control the opacity of the rows.

For example,

```
let chart = new soda.Chart({
  selector: "div#soda-chart",
  rowColors: ["cyan", "green", "purple"],
  rowOpacity: 0.5,
});

chart.render({ rowCount: 10 });
```

will produce something like:

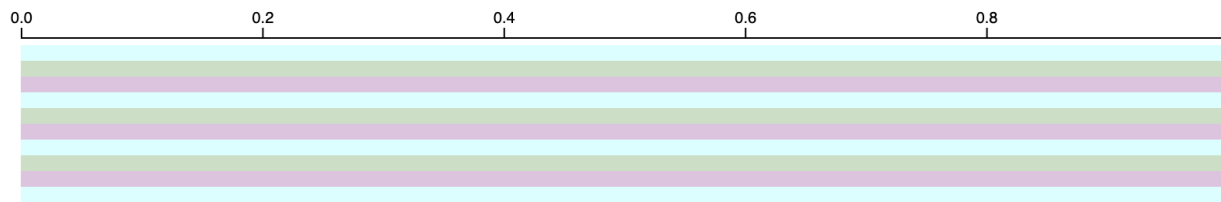


Fig. 3: A blank Chart set up with row colors cyan, green, and purple.

## Zooming and panning

Charts can be configured to enable zooming and panning by setting the `zoomable` property to true on the *ChartConfig*.

```
let chart = new Chart({
  selector: "div#soda-chart",
  zoomable: true
});
```

Zoomable Charts may be zoomed with ctrl + scrolling and panned by clicking and dragging. Any glyphs added to the Chart using SODA's rendering API will respond appropriately to zooming and panning events. Any SVG elements that are added to the Chart by other means will remain unaltered.

Zooming may be constrained with the `zoomConstraint` property, which is a tuple that bounds the scaling factor.

For example,

```
let chart = new Chart({
  selector: "div#soda-chart",
  zoomable: true,
  zoomConstraint: [1, 100]
});
```

would prevent the chart from being zoomed out from the point it's initially rendered at, and would allow zooming in by a factor of 100.

Panning may be constrained with the `domainConstraint` property, which is a callback function that returns the desired extent of the domain. The callback function receives the Chart itself as an argument.

For example:

```
let chart = new Chart({
  selector: "div#soda-chart",
  zoomable: true,
  // chart.initialDomain is the extent of the
  // domain set during the last render call
  domainConstraint: (chart) => chart.initialDomain
});
```

would prevent the Chart from being panned outside of the domain set by the last *render()* call.

## Resizing

Charts can be configured to automatically resize themselves as their DOM container resizes by setting the *resizable* property to true on the *ChartConfig*.

```
let chart = new Chart({
  selector: "div#soda-chart",
  resizable: true
});
```

If this is enabled, when the Chart's container is resized (e.g. when the browser window is resized), the Chart will re-render itself to display the same domain in the new range. As with zooming, any glyphs rendered with SODA will be affected, but any SVG elements added by other means will remain unaffected.

## Zoom and resize callbacks

You can optionally supply both a *postZoom* and a *postResize* callback in the *ChartConfig*, which will be called after zoom/pan events and resize events, respectively.

For example:

```
let chart = new Chart({
  selector: "div#soda-chart",
  zoomable: true,
  resizable: true,
  postZoom() {
    console.log(this, "zoomed!");
  },
  postResize() {
    console.log(this, "resized!");
  }
});
```

### 1.5.3 Chart scales

To help position glyphs in the viewport, Charts maintain a couple of scale functions.

The first is the *xScale*, which maps from semantic coordinates (e.g. positions in a sequence) to pixel x-coordinates relative to the origin of the viewports. The *xScale* is used extensively by the defaults in the *rendering* API, and also in the zooming, panning, and resizing logic.

The second is the *yScale*, which maps row numbers to the pixel y-coordinates that delineate each of the conceptual rows in the Chart's viewport.

### 1.5.4 Chart observers

The Chart observer is a SODA pattern in which an object can respond to changes in a Chart. The pattern is currently not very fleshed out, and it is currently only used by the *ZoomSyncer* object, which synchronizes the zoom level across multiple Charts. At some point, we will overhaul this system to make it much more useful, but you may find some use from it in its current state.

You can create an object that extends the abstract class *ChartObserver*, add Charts to it, and then configure Charts to call *Chart.alertObservers()*.

## 1.6 Loading annotation data

Before jumping into building a SODA visualization, you'll probably want to consider how you're going to get your data loaded into the browser in a form that SODA is happy with. On this page, we'll cover just that.

For a high-level overview of the process, you can refer to this flowchart:

### 1.6.1 Using a SODA-tailored REST API

In an ideal world for SODA, your data source would have two qualities:

1. The data is stored in a database with an accompanying REST API
2. The REST API returns JSONs that may be serialized directly into objects that implement the *Annotation* interface

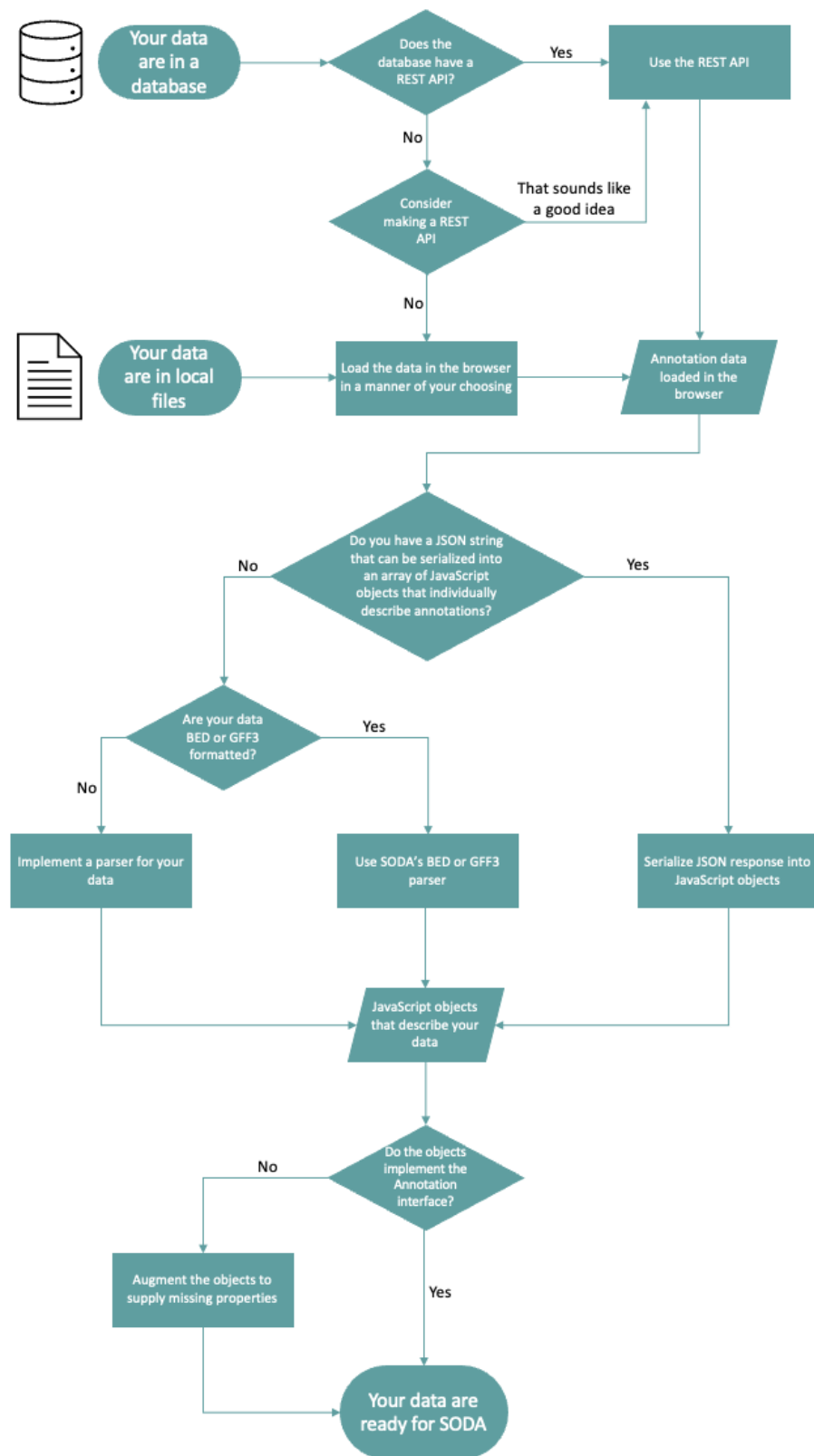
If you can configure a database in this fashion, you can load data into a SODA visualization with very little effort.

#### Example SODA-tailored REST API

We host a toy database that we use to produce several of the visualizations in the *Examples* section. The database has a REST API that exposes the endpoint <https://sodaviz.org/data/examples/default>, which returns a JSON string describing Dfam annotations:

```
// the response of a GET request to our toy database
[
  {
    "id": "dfam-nrph-1", // <- this satisfies the id property
    "start": 10464,      // <- this satisfies the start property
    "end": 10954,        // <- this satisfies the end property
    "strand": "-",
```

(continues on next page)



(continued from previous page)

```

    "family": "TAR1",
    "evaluate": 1.1E-103,
    "divergence": 10.26
  },
  {
    "id": "dfam-nrph-2",
    "start": 10826,
    "end": 11463,
    "strand": "-",
    "family": "TAR1",
    "evaluate": 4.1E-165,
    "divergence": 8.78
  },
  ...
]

```

## Fetch requests and object serialization

One of the easiest ways to perform a GET request in the browser is to use the JavaScript [fetch API](#). If the response to your GET request is a JSON string, it can be automatically serialized into JavaScript objects.

For example, we might load the JSON from the above section with the following code:

```

// an example fetch request and subsequent JSON serialization

fetch("https://sodaviz.org/data/examples/default")
  .then((response) => response.json())
  .then((annotations: soda.Annotation[]) => doSomething(annotations));
                                // ^^ you would replace doSomething()
                                //    with a SODA rendering API call

```

## Defining an interface for your data

The code in the previous section will certainly produce objects that SODA can use, but it has a small problem: we haven't told TypeScript that these objects have data properties other than `id`, `start`, and `end`. If we want to make use of these properties to influence our visualization, we'll need to write a simple interface that better describes our data:

```

interface DfamAnnotation extends soda.Annotation { // <- we get id, start, and
  strand: string;                                //    end for free if we
  family: string;                                //    extend soda.Annotation
  evaluate: number; // <- the property names and types
  divergence: number; //    need to match the JSON fields
}

fetch("https://sodaviz.org/data/examples/default")
  .then((response) => response.json())
  .then((annotations: soda.DfamAnnotation[]) => doSomething(annotations));
                                // ^^ now we can tell the TypeScript compiler
                                //    that the objects we serialized are described
                                //    by the DfamAnnotation interface

```

---

**Note:** If you're using JavaScript rather than TypeScript, you don't need to write interfaces to provide type information.

---

## 1.6.2 Using a standard REST API

If you're planning to use a public database resource (e.g. UCSC, Ensembl, etc.), it's unlikely that the REST API will be implemented in the quite the same manner described in the previous section. However, they tend to respond in a format that can be easily coerced into a format that makes SODA happy.

In this section, we'll look at how you might load annotation data using the [UCSC Genome Browser API](#) using data from the [Pfam in GENCODE track](#).

### Format a GET request

The first step is to figure out which REST API endpoint will return the data you're interested in. For example, the UCSC genome browser API has an endpoint that returns annotations for a chosen annotation track, which is at the url: <https://api.genome.ucsc.edu/getData/track>.

Digging into the UCSC API documentation a bit, we'll discover that this endpoint has a handful of aptly named parameters:

- **track** - the annotation track we want data from
- **genome** - the genome that we want annotations for
- **chrom** - the chromosome that we want annotations for
- **start** - the start coordinate of our query in base pairs
- **end** - the end coordinate of our query in base pairs

With this information in hand, we can craft a GET request, which might look something like:

```
https://api.genome.ucsc.edu/getData/track?genome=hg38;track=ucscGenePfam;chrom=chr1;start=0;end=1000000
```

### Determine the response structure

Now we need to get a sense of what the response looks like. An easy way to figure out how responses are structured is to just paste the link in your browser, which should dump the text onto your screen. You might find it useful to paste that text in a [JSON formatter](#) to make it a bit easier to digest.

The response to the above query looks like:

```
{
  "downloadTime": "2022-08:25T19:44:29Z", // <-
  "downloadTimeStamp": 1661456669,      // <-
  "genome": "hg38",                     // <-
  "dateTime": "2022-05-15T13:39:52",     // <- you'll often find lots of
  "dateTimeStamp": 1652647192,           // <- metadata that comes along
  "trackType": "bed 12",                 // <- for the ride. we are just
  "track": "ucscGenePfam",               // <- going to ignore all of this
  "start": 0,                           // <-
```

(continues on next page)



(continued from previous page)

```

"end":100000,           // <-
"chrom":"chr1",         // <-
"ucscGenePfam":[ // <- this is the data we are after
  {
    "bin":585,
    "chrom":"chr1",
    "chromStart":69168,
    "chromEnd":69969,
    "name":"7tm_4",
    "score":0,
    "strand":"+ ",
    "thickStart":0,
    "thickEnd":0,
    "reserved":0,
    "blockCount":1,
    "blockSizes":"801,",
    "chromStarts":"0,"
  },
  ... // <- for the sake of brevity, we've
      // removed some members of this array
],
"itemsReturned":3
}

```

You'll probably notice that the data we're interested in is stored on the `ucscGenePfam` property of the object.

## Defining interfaces for responses

Once we've got the response structure figured out, we can start crafting our data-loading code, starting with a couple interfaces:

```

// first we'll describe the individual annotation records
interface PfamRecord {
  chrom: string;           // <- we'll specify only the properties we actually
  chromStart: number;      // <- care about. the rest of them will be there
  chromEnd: number;        // <- regardless, but we don't need to tell that to
  score: number;           // <- to the TypeScript compiler
  strand: string;          // <-
}

// now we can precisely describe the location and
// type of the annotation records in the response
interface UcscResponse {
  ucscGenePfam: PfamRecord[]
}

```

## Producing proper Annotation objects

Now we can serialize a response into objects, but we need a way to shape them into a form that SODA is happy to work with.

Our PfamRecord interface fails to comply with the *Annotation* interface in two ways: (i) we've got the `start` and `end` properties, but they are named `chromStart` and `chromEnd`; and (ii) we don't have the `id` property, and we don't actually have any data that could be used for it. Here, we'll look at a couple strategies to fix these issues.

## Using augment()

One approach is to use SODA's *augment* function. In short, `augment()` takes a list of any JavaScript objects and some instructions on how to transform them into objects that comply with the *Annotation* interface. There's a subtle but important nuance to using `augment()`: if you set the virtual flag to `true` on a property, the callback function will be evaluated and the value will be added as a real property on the object, but if you omit `virtual` or set it to `false`, the callback function will be added as a *getter* for that property.

**Note:** This code block starts to dip into some of TypeScript's more nuanced language features, namely *intersection types* and *generics*.

```
// PfamAnnotation is a type alias for the type intersection of
// the PfamRecord and soda.Annotation types
type PfamAnnotation = PfamRecord & soda.Annotation

fetch(
  "https://api.genome.ucsc.edu/getData/track?genome=hg38;track=ucscGenePfam;chrom=chr1;
  ↪start=0;end=1000000"
)
.then((response) => response.json())
.then((response) => response.json())
.then((response: UcscResponse) => response.ucscGenePfam)
.then((records: PfamRecord[]) =>
  soda.augment<PfamRecord>({ // <- augment has a generic type parameter R, and
    objects: records,       // it returns the type (R & soda.Annotation)
    id: { fn: () => soda.generateId("pfam") }, // <- real property
    start: { fn: (r) => r.chromStart, virtual: true }, // <- getter
    end: { fn: (r) => r.chromEnd, virtual: true }, // <- getter
  })
)
.then((annotations: PfamAnnotation[]) => doSomething(annotations));
```

## Writing a class

If you want a bit more control over your objects, you can write a class instead. That might look something like:

```
class PfamAnnotation implements soda.Annotation {
  id: string;
  start: number;
  end: number;
  score: number;
```

(continues on next page)

(continued from previous page)

```

strand: string;

constructor(record: PfamRecord) {
  this.id = soda.generateId("pfam");
  this.start = record.chromStart;
  this.end = record.chromEnd;
  this.score = record.score;
  this.strand = record.strand;
}
}
fetch(
  "https://api.genome.ucsc.edu/getData/track?genome=hg38;track=ucscGenePfam;chrom=chr1;
  ↪start=0;end=1000000"
)
  .then((response) => response.json())
  .then((response: UcscResponse) => response.ucscGenePfam)
  .then((records: PfamRecord[]) => records.map((r) => new PfamAnnotation(r)))
  .then((annotations: PfamAnnotation[]) => {
    console.log(annotations);
    chart.render({ annotations });
  });
}

```

### 1.6.3 Local files

If you are interested in loading local files into SODA, you'll need to come up with a way to load the file into the browser, then come up with a way to parse it into suitable objects. In this section we'll give a simple example on how to accomplish that.

#### Setting up a webpage to accept a file input

To start off, you might add a file input tag to your page HTML:

```

// hypothetical index.html

<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>soda viz</title>
  </head>
  <body>
    <div class="soda-chart">           // <- your visualization would live here
    <input type="file" id="file-input"> // <- here's your file input
    <button type="button" id="submit"> // <- here's a button we can set up to
      Submit                          //   submit the file
    </button>
  </body>
<script src="main.js"></script>      // <- this would point to your
</html>                              //   data loading + SODA code

```

Next, you might write some TypeScript that loads your data files:

```
// hypothetical main.js

function loadData() {
  let input = <HTMLInputElement>document.getElementById("file-input")!;
  let file = input.files![0];
  file.text()
    .then((data: string) => submitData(data)); // <- submitData() would be
                                              // responsible for parsing
                                              // the file and so on
}

document.getElementById("submit")!
  .addEventListener("click", loadData);
```

This gets us the file loaded into browser memory as a string. Moving on, we'll need to decide how we're going to process it into objects. The approach we take will, unsurprisingly, depend on how your file is formatted.

## JSON files

If the file happens to be a JSON string that describes record objects, you can employ the strategies presented in the above sections concerning REST APIs. We'll note that if your data files are products of your own tools, you might want to format your output as a SODA-tailored JSON so that you can simplify the loading process.

## BED or GFF3 files

If the file is BED or GFF3 formatted, you can use SODA's parsers to produce objects. For example, let's say you loaded a the following BED data into the browser as a string:

chr7	127471196	127472363	Pos1	0	+	127471196	127472363	255,0,0
chr7	127472363	127473530	Pos2	0	+	127472363	127473530	255,0,0
chr7	127473530	127474697	Pos3	0	+	127473530	127474697	255,0,0
chr7	127474697	127475864	Pos4	0	+	127474697	127475864	255,0,0
chr7	127475864	127477031	Neg1	0	-	127475864	127477031	0,0,255
chr7	127477031	127478198	Neg2	0	-	127477031	127478198	0,0,255
chr7	127478198	127479365	Neg3	0	-	127478198	127479365	0,0,255
chr7	127479365	127480532	Pos5	0	+	127479365	127480532	255,0,0
chr7	127480532	127481699	Neg4	0	-	127480532	127481699	0,0,255

You can parse it into *BedAnnotation* objects like:

```
// suppose this is our submitData() function from above
function submitData(data: string) {
  let annotations: soda.BedAnnotation[] = soda.parseBedRecords(data);
  doSomething(annotations) // <- again, you'd probably be replacing
                          // doSomething() with SODA rendering calls
}
```

## Anything else

If your data are in some other format, you're more or less left to your own devices. Unless you have something remarkably strange, the process will probably end up being quite similar to the strategies described on this page.

If you think there's a case to be made for adding more parsers to the SODA API, we'd be interested in [hearing about it](#).

## 1.7 Rendering

SODA's rendering API takes *Annotation* objects and represents them as SVG glyphs inside of *Chart* viewports in the DOM. This section explains the ins and out of that process.

### 1.7.1 Glyph rendering functions

To render glyphs, you simply make a call to a glyph rendering function with an appropriate *GlyphConfig* object. The properties on the config object describe which *Annotation* objects to use, where to render the glyphs, and how to style the glyphs. Each glyph rendering function has a corresponding config defined by an interface. For example, a simple call to the *rectangle* function with a minimally specified *RectangleConfig* may look like:

```
rectangle({
  chart: chart,           // "chart" is a Chart object
  annotations: annotations, // "annotations" is an array of Annotation objects
});
```

### Glyph selectors

The *selector* property in a *GlyphConfig* is a string used to help differentiate between distinct collections of glyphs that have been rendered using the same set of *Annotation* objects in the same *Chart*. SODA applies the selector as a CSS class on the DOM elements that make up the glyphs.

The selector property also allows a nuanced feature: if you use the same selector string in a subsequent call to a glyph rendering function, the glyphs from the prior call using that selector will be replaced.

### Glyph properties

The properties in a *GlyphConfig* that control glyph styling are typed as *GlyphProperty*, which is simply an alias of the type union of a static value and a *GlyphCallback*. A *GlyphCallback* is another type alias for a simple callback function that takes an *AnnotationDatum* as the sole argument and returns a value.

For example, if we were to add static *GlyphProperty* to a *rectangle()* call, it might look like:

```
rectangle({
  chart: chart,
  annotations: annotations,
  fillColor: "red",
  fillOpacity: 0.5
});
```

To illustrate how to take full advantage of the flexibility of *GlyphProperties*, imagine we were using a custom *Annotation* data type:

```
interface CustomAnnotation implements Annotation {
  id: string;      // <- the fields required by Annotation
  start: number;
  end: number;
  color: string;   // <- our custom fields
  score: number;
}
```

Then, we could use callback `GlyphProperties` like:

```
// explicit type parameters have been added here for clarity, but
// the TypeScript compiler is usually smart enough to infer them
rectangle<CustomAnnotation, Chart<RenderParams>>({
  chart: chart,
  annotations: annotations,
  fillColor: (d: AnnotationDatum<CustomAnnotation, Chart<RenderParams>>) =>
    d.a.color,
  fillOpacity: (d: AnnotationDatum<CustomAnnotation, Chart<RenderParams>>) =>
    d.a.score
});
```

Check out the examples section to see more examples.

## 1.7.2 The canonical rendering pattern

In SODA, the canonical rendering pattern is to define a rendering routine inside of a `Chart` object. The rendering routine described here is a pattern that we find straightforward, but it is by no means the only way to achieve a visualization. Once you know a bit about how SODA works, you should find it pretty easy to extend the `Chart` class and assume greater control over the fine details of rendering process.

### Default rendering routine

The default rendering routine is broken up into several steps, which will be described in the following sections.

### RenderParams

The `RenderParams` is an object that is passed as the sole argument into `Chart.render()`. The default `RenderParams` implementation looks like:

```
interface RenderParams {
  annotations?: Annotation[]; //<- the list of Annotation objects to render
  start?: number;             //<- the start of the interval to be rendered
  end?: number;               //<- the end of the interval to be rendered
  rowCount?: number;          //<- fix the height of the chart to a number of rows
}
```

You'll notice that every property on the interface is optional. This means you can think of the default `RenderParams` implementation as something of a suggestion. However, the default rendering routine is set up to respond to the presence of each of the properties in this implementation. With that in mind, you may find some use in adapting or extending the default `RenderParams`.

### Chart.render()

The render() method calls each of the configurable rendering callbacks in succession. Each of the callbacks receives the RenderParams object as an argument. The callbacks can be overwritten in the *ChartConfig* or reassigned at runtime.

```
public render(params: P): void {
  this.renderParams = params;
  this.updateLayout(params);
  this.updateRowCount(params);
  this.updateDimensions(params);
  this.updateDomain(params);
  this.draw(params);
  this.postRender(params);
}
```

### Chart.updateLayout()

The updateLayout() callback is responsible for producing a *VerticalLayout* for the Chart. By default, the rendering API uses the Chart's layout object to vertically position glyphs into rows. By passing a list of Annotation objects into one of SODA's layout functions, a VerticalLayout that guarantees no horizontal overlap will be produced.

The default updateLayout method looks like:

```
public defaultUpdateLayout(params: P): void {
  if (params.annotations != undefined) {
    this.layout = intervalGraphLayout(params.annotations);
  }
}
```

### Chart.updateRowCount()

The updateRowCount() callback is responsible for updating the Chart's rowCount property. A handful of methods use the rowCount property to properly adjust the heights of the Chart's DOM elements.

The default updateRowCount method looks like:

```
public defaultUpdateRowCount(params: P): void {
  this.rowCount =
    params.rowCount != undefined ? params.rowCount : this.layout.rowCount;
}
```

### Chart.updateDimensions()

The updateDimensions() callback is responsible for updating the Chart's DOM element dimensions to accommodate the render. By default, only the Chart's vertical dimensions are adjusting during a render call, and it is assumed that the rowCount is properly set before the method is called.

The default updateDimensions method looks like:

```
public defaultUpdateDimensions(params: P): void {
  this.updateDivHeight();
  this.updatePadHeight();
  this.updateViewportHeight();
}
```

### Chart.updateDomain()

The `updateDomain()` callback is responsible for updating the Chart's domain. This effectively controls the interval that is initially displayed after the render call finishes. Adjusting the domain can be thought of as applying zooming or panning on the Chart's viewport.

The default `updateDomain` method looks like:

```
public defaultUpdateDomain(params: P): void {
  let domain = this.domain;
  if (params.start !== undefined && params.end !== undefined) {
    domain = [params.start, params.end];
  } else if (params.annotations !== undefined) {
    domain = Chart.getDomainFromAnnotations(params.annotations);
  }
  this.initialDomain = domain;
  this.domain = domain;
}
```

### Chart.draw()

The `draw()` callback is responsible for using the rendering API to place glyphs in the Chart. The default implementation calls `Chart.addAxis()` and renders the annotations as rectangle glyphs.

The default `draw` method looks like:

```
public defaultDraw(params: P): void {
  this.addAxis();
  rectangle({
    chart: this,
    annotations: params.annotations || [],
    selector: "soda-rect"
  });
}
```

### Customizing the rendering routine

In building your own SODA visualization, most of the work is likely to be in customizing the `draw()` rendering callback. The default `draw()` produces a lackluster display of black rectangle glyphs. If you wanted to add some color, you could do something like this when you instantiate your Chart:

```
let chart = new Chart({
  selector: "div#soda-chart",
  draw(this, params) {
```

(continues on next page)



(continued from previous page)

```

    this.addAxis()
    rectangle({
      chart: this,
      annotations: params.annotations || [],
      selector: "soda-rect",
      fillColor: "cyan" // <- this will make the rectangle glyphs cyan
    })
  }
});

```

Understanding the nuances of customizing the rendering routine is probably best learned by example, so check out the examples section to learn more.

### 1.7.3 Interactivity

SODA allows you to define callback functions that are called whenever a glyph is clicked or hovered. The callback functions are loosely typed by `InteractionCallback`. The `InteractionCallback` type serves as an indicator of the arguments SODA will pass to your callback function when it is executed:

```

type InteractionCallback<A extends Annotation, C extends Chart<any>> = {
  (
    s: d3.Selection<any, AnnotationDatum<A, C>, any, any>, // <- a D3 selection to the
    ↪ glyph's DOM element
    d: AnnotationDatum<A, C>                               // <- a reference to the
    ↪ Annotation object and the Chart
  ): void;
};

```

These arguments are passed in by default, and you are free to arbitrarily define the function body. If you already know a bit about D3 (or are willing to learn), you can use the `Selection` argument to modify the glyph in the DOM. With the `AnnotationDatum` argument, you gain access to the `Annotation` that the glyph was rendered with and the `Chart` that it is rendered in.

The interaction API is similar to the glyph rendering API: you simply make a call to an interaction function with an appropriate `InteractionConfig` object. For example, a simple call to the `clickBehavior` function with `ClickConfig` may look like:

```

clickBehavior({
  annotations: annotations, // <- "annotations" is an array of Annotation objects
  click: (s, d) => {        // <- "click" is applied
    alert(`${d.a.id} clicked`)
  }
});

```

### 1.7.4 Glyph mapping

Internally, SODA maps Annotation objects to the glyphs that they have been used to render. Specifically, keys are built using the *id* property of the Annotation object, the *selector* used in the rendering call, and the *id* property of the target Chart. The mapping information can be accessed with the *queryGlyphMap* function, which returns D3 selections of the DOM elements that make up the glyphs. You can optionally specify any number of the components of the keys to build a query, effectively changing the granularity of the search.

Calls to the *queryGlyphMap* function may look like:

```
// this will return a single glyph
let specificGlyph = queryGlyphMap({
  id: "ann-1",
  chart: chart,
  selector: "gene-rectangles",
})

// this will return all of the glyphs in "chart"
// rendered with the selector: "gene-rectangles"
let rectanglesInChart = queryGlyphMap({
  chart: chart,
  selector: "gene-rectangles"
})

// this will return all of the glyphs in every Chart
// rendered with the selector: "gene-rectangles"
let allRectangles = queryGlyphMap({
  selector: "gene-rectangles"
})

// this will return all of the glyphs in "chart"
let allInChart = queryGlyphMap({
  chart: chart,
})

// this will return every glyph in every Chart
let allGlyphs = queryGlyphMap({})
```

This guide explains SODA concepts and serves as a starting point for understanding how SODA functions at a high level. If you find something that seems missing or poorly explained, we're interested in [hearing about it](#)

## EXAMPLES

This page shows some examples of simple SODA visualizations. Each example has an embedded [CodePen](#), which can be edited live by selecting the “TypeScript” tab.

The data from each example is loaded with a `fetch` call to the Sodaviz example database API, which returns JSON strings that serialize directly into JavaScript objects that implement the *Annotation* interface.

For example, creating a chart and rendering data from a fetch request looks something like:

```
// create a chart in the div with the ID "soda-chart"
let chart = new soda.Chart({ selector: "div#soda-chart" });

// fetch the example data from this endpoint
fetch("https://sodaviz.org/data/examples/default")
  // when we get the response, call json() to serialize it into objects
  .then((response: Response) => response.json())
  // when we have our objects, render them in the Chart
  .then((annotations: soda.Annotation[]) => {
    chart.render({ annotations: annotations })
  });
```

For the sake of brevity, the following examples omit some syntax, which leaves them looking a little more like:

```
let chart = new soda.Chart({ selector: "div#soda-chart" });

fetch("https://sodaviz.org/data/examples/default")
  .then(response => response.json())
  .then(annotations => chart.render({ annotations }));
```

---

**Tip:** Each example can be zoomed with ctrl + scroll wheel and panned by clicking and dragging.

---

## 2.1 Simple rectangles

```
import * as soda from "https://cdn.skypack.dev/@sodaviz/soda@0.12.0";

let chart = new soda.Chart({
  selector: "div#soda-chart",
  zoomable: true,
  resizable: true
});

fetch("https://sodaviz.org/data/examples/default")
  .then(response => response.json())
  .then(annotations => chart.render({ annotations }));
```

---

## 2.2 Styled rectangles

```
import * as soda from "https://cdn.skypack.dev/@sodaviz/soda@0.12.0";
import * as d3 from "https://cdn.skypack.dev/d3@7.4.4";

// we'll use this D3 color scale later to easily pick colors
let colors = d3.scaleOrdinal(d3.schemeTableau10);

// we'll define a custom Annotation interface that
// describes some extra fields that exist in our records
interface CustomAnnotation extends soda.Annotation {
  family: string;
  divergence: number;
}

// we'll define a custom RenderParams interface that
// describes our custom render data payload
interface CustomRenderParams extends soda.RenderParams {
  annotations: CustomAnnotation[];
}

// we'll explicitly type our Chart with our CustomRenderParams
// so that the TypeScript compiler knows what we're intending
let chart = new soda.Chart<CustomRenderParams>({
  selector: "div#soda-chart",
  zoomable: true,
  resizable: true,
  // we'll write a custom draw() callback to overwrite
  // the default rendering behavior of the Chart
  draw(params) {
    this.addAxis();
    soda.rectangle({
      chart: this,
      annotations: params.annotations,
      // these callbacks will be evaluated for each glyph
    });
  }
});
```

(continues on next page)

(continued from previous page)

```

        fillColor: (d) => colors(d.a.id),
        fillOpacity: (d) => (100 - d.a.divergence) / 100,
        strokeColor: "none"
    });
  }
});

fetch("https://sodaviz.org/data/examples/default")
  .then(response => response.json())
  .then(annotations => chart.render({ annotations }));

```

## 2.3 Dynamic text

```

import * as soda from "https://cdn.skypack.dev/@sodaviz/soda@0.12.0";
import * as d3 from "https://cdn.skypack.dev/d3@7.4.4";

let colors = d3.scaleOrdinal(d3.schemeTableau10);

interface CustomAnnotation extends soda.Annotation {
  family: string;
}

interface CustomRenderParams extends soda.RenderParams {
  annotations: CustomAnnotation[];
}

let chart = new soda.Chart<CustomRenderParams>({
  selector: "div#soda-chart",
  rowHeight: 14,
  zoomable: true,
  resizable: true,
  draw(params) {
    this.addAxis();
    soda.rectangle({
      chart: this,
      annotations: params.annotations,
      fillColor: (d) => colors(d.a.id),
      strokeColor: "none"
    });
    // we'll call dynamicText() with the same Annotations
    // that we're using to render the rectangles
    soda.dynamicText({
      chart: this,
      annotations: params.annotations,
      // the dynamic text glyph displays the longest string that will
      // fit into the space it has available in the viewport
      text: (d) => [`${d.a.family} - ${d.a.id}`, d.a.family, "..."]
    });
  }
});

```

(continues on next page)

(continued from previous page)

```

    }
  });

  fetch("https://sodaviz.org/data/examples/default")
    .then((response) => response.json())
    .then((annotations) => chart.render({ annotations }));

```

## 2.4 Interactivity

```

import * as soda from "https://cdn.skypack.dev/@sodaviz/soda@0.12.0";
import * as d3 from "https://cdn.skypack.dev/d3@7.4.4";

let colors = d3.scaleOrdinal(d3.schemeTableau10);

interface CustomAnnotation extends soda.Annotation {
  family: string;
}

interface CustomRenderParams extends soda.RenderParams {
  annotations: CustomAnnotation[];
}

let chart = new soda.Chart<CustomRenderParams>({
  selector: "div#soda-chart",
  rowHeight: 20,
  zoomable: true,
  resizable: true,
  // we'll write a simple draw() callback that
  // gives us colored rectangles
  draw(params) {
    this.addAxis();
    soda.rectangle({
      chart: this,
      annotations: params.annotations,
      fillColor: (d) => colors(d.a.id),
      strokeColor: "none"
    });
  },
  // now we'll write a postRender() callback that
  // applies some interactions to the glyphs
  postRender(params) {
    soda.clickBehavior({
      chart: this,
      annotations: params.annotations,
      // this function is evaluated when a glyph is clicked
      click: (
        // s is a d3 Selection of the glyph in the DOM
        s: d3.Selection<any, any, any, any>,

```

(continues on next page)

(continued from previous page)

```

    // d is the AnnotationDatum bound to the glyph
    d: soda.AnnotationDatum<CustomAnnotation, Chart<CustomRenderParams>>
  ) => alert(`${d.a.id} clicked`)
});
soda.hoverBehavior({
  chart: this,
  annotations: params.annotations,
  // this function is evaluated when a glyph is moused over
  mouseover: (s, d) => s.style("stroke", "black"),
  // this function is evaluated when a glyph is no longer moused over
  mouseout: (s, d) => s.style("stroke", "none")
});
soda.tooltip({
  chart: this,
  annotations: params.annotations,
  text: (d) => d.a.family
});
}
});

fetch("https://sodaviz.org/data/examples/default")
  .then((response) => response.json())
  .then((annotations) => chart.render({ annotations }));

```

## 2.5 Plot annotations

```

import * as soda from "https://cdn.skypack.dev/@sodaviz/soda@0.12.0";

interface CustomRenderParams extends soda.RenderParams {
  annotations: PlotAnnotation[];
}

let chart = new soda.Chart<CustomRenderParams>({
  selector: "div#soda-chart",
  zoomable: true,
  resizable: true,
  rowHeight: 100,
  draw(params) {
    this.addAxis();
    soda.linePlot({
      chart: this,
      annotations: params.annotations
    });
    soda.verticalAxis({
      chart: this,
      // setting this allows the axes to overflow into
      // the SVG pad, preventing them from clipping
      target: soda.BindTarget.Overflow,

```

(continues on next page)

(continued from previous page)

```
        annotations: params.annotations
    });
}
});

fetch("https://sodaviz.org/data/examples/plots")
    .then((response) => response.json())
    .then((annotations) => chart.render({ annotations }));
```

---

## 2.6 Sequence annotations

```
import * as soda from "https://cdn.skypack.dev/@sodaviz/soda@0.12.0";

interface CustomRenderParams extends soda.RenderParams {
    annotations: SequenceAnnotation[];
}

let chart = new soda.Chart<CustomRenderParams>({
    selector: "div#soda-chart",
    zoomable: true,
    resizable: true,
    draw(params) {
        this.addAxis();
        soda.sequence({
            chart: this,
            annotations: params.annotations
        });
    }
});

fetch("https://sodaviz.org/data/examples/sequence")
    .then((response) => response.json())
    .then((annotations) => chart.render({ annotations }));
```



## 3.1 Classes

### 3.1.1 AnnotationGroup

```
class AnnotationGroup<A extends Annotation>
```

An Annotation class that contains a group of Annotations.

#### Type parameters

- A: Annotation

#### Constructors

```
(config: AnnotationGroupConfig <A>): AnnotationGroup
```

#### Type parameters

- A: Annotation

#### Parameters

- config: AnnotationGroupConfig

#### Properties

##### annotations

```
annotations: A []
```

The group of Annotations that live in this object.

## end

```
end: number
```

The end coordinate of the Annotation.

## id

```
id: string
```

A unique identifier for the Annotation.

## start

```
start: number
```

The start coordinate of the Annotation.

## Methods

### add

```
add(ann: A | A []): void
```

Add an Annotation or list of Annotations to the group.

#### Parameters

- ann: **A** | **A** []

**Returns:** void

### addAnnotation

```
addAnnotation(ann: A): void
```

Add an Annotation to the group.

#### Parameters

- ann: **A**

**Returns:** void

### 3.1.2 Chart

```
class Chart<P extends RenderParams>
```

This is used to render Annotation objects as glyphs in the browser.

#### Type parameters

- P: RenderParams

#### Constructors

```
(config: ChartConfig <P>): Chart
```

#### Type parameters

- P: RenderParams

#### Parameters

- config: ChartConfig

#### Properties

##### **\_divHeight**

```
_divHeight: undefined | string
```

The CSS height property of the Chart's div.

##### **\_divMargin**

```
_divMargin: undefined | string | number
```

The CSS margin property of the Chart's div.

##### **\_divOutline**

```
_divOutline: undefined | string
```

The CSS outline property of the Chart's div.

### **`_divOverflowX`**

`_divOverflowX`: **undefined** | **string**

The CSS overflow-x property of the Chart's div.

### **`_divOverflowY`**

`_divOverflowY`: **undefined** | **string**

The CSS overflow-y property of the Chart's div.

### **`_divWidth`**

`_divWidth`: **undefined** | **string**

The CSS width property of the Chart's div.

### **`_padHeight`**

`_padHeight`: **undefined** | **string**

The stored value of the pad SVG height property.

### **`_padWidth`**

`_padWidth`: **undefined** | **string**

The stored value of the pad SVG width property.

### **`_renderParams`**

`_renderParams`: **undefined** | **P**

The last used render parameters.

### **`_transform`**

`_transform`: **Transform**

The Transform object that describes the current zoom transformation.

### **\_viewportHeight**

```
_viewportHeight: undefined | string
```

The stored value of the viewport SVG height property.

### **\_viewportWidth**

```
_viewportWidth: undefined | string
```

The stored value of the viewport SVG width property.

### **containerSelection**

```
containerSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's DOM container. This is a pre-existing DOM element (probably a div).

### **defSelection**

```
defSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's defs element. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Element/defs>

### **divSelection**

```
divSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's div container. This is created when the Chart is instantiated and placed inside of the selected container in the DOM.

### **domainConstraint**

```
domainConstraint: (chart: Chart <P>): None
```

This constrains the Chart's domain, which in turn constrains both zoom level and panning. The parameter is a callback function that is evaluated after each zoom event to produce an interval that constrains the domain.

**draw**

draw: (params: **P**): **void**

The rendering callback that should be responsible for drawing glyphs with the rendering API.

**glyphModifiers**

glyphModifiers: **GlyphModifier** <any, any> []

A list of GlyphModifiers that control the glyphs rendered in the Chart.

**highlightSelection**

highlightSelection: **Selection** <any, any, any, any>

A d3 selection of the Chart's highlight.

**id**

id: **string**

A unique identifier for the Chart.

**initialDomain**

initialDomain: **None**

The initialized domain of the Chart when render() is called with the initializeXScale flag.

**layout**

layout: **VerticalLayout**

**leftPadSize**

leftPadSize: **number**

The number of pixels of padding on the left side of the Chart.

### lowerPadSize

```
lowerPadSize: number
```

The number of pixels of padding on the bottom of the Chart.

### observers

```
observers: ChartObserver []
```

A list of observers attached to the Chart.

### overflowViewportSelection

```
overflowViewportSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's viewport that allows rendering overflow.

### padSelection

```
padSelection: Selection <any, any, any, any>
```

A d3 selection of the viewport's padding container.

### padSize

```
padSize: number
```

The number of pixels of padding around each edge of the Chart.

### postRender

```
postRender: (params: P): void
```

The callback function that the Chart executes after render() is called.

### postResize

```
postResize: (): void
```

The callback function that the Chart executes after resize() is called.

**postZoom**

```
postZoom: () : void
```

The callback function that the Chart executes after zoom() is called.

**resizable**

```
resizable: boolean
```

This controls whether or not the Chart has automatic resizing enabled.

**rightPadSize**

```
rightPadSize: number
```

The number of pixels of padding on the right side of the Chart.

**rowColors**

```
rowColors: undefined | string []
```

A list of colors that will color the Chart's rows in a repeating pattern.

**rowCount**

```
rowCount: number
```

The number of rows in the Chart.

**rowHeight**

```
rowHeight: number
```

The height in pixels of a horizontal row in the Chart. This defaults to a value of 10.

**rowOpacity**

```
rowOpacity: number
```

The opacity of the colored row stripes.



### selector

```
selector: string
```

A string that can be used to uniquely select the target DOM container.

### updateDimensions

```
updateDimensions: (params: P): void
```

The rendering callback function that should be responsible for updating the Chart's DOM element dimensions.

### updateDomain

```
updateDomain: (params: P): void
```

The rendering callback function that should be responsible for updating the domain of the Chart.xScale property.

### updateLayout

```
updateLayout: (params: P): void
```

The rendering callback function that should be responsible for updating the Chart.layout property.

### updateRowCount

```
updateRowCount: (params: P): void
```

The rendering callback function that should be responsible for updating the Chart.rowCount property.

### upperPadSize

```
upperPadSize: number
```

The number of pixels of padding on the top of the Chart.

### viewportHeightPx

```
viewportHeightPx: number
```

The stored height of the viewport SVG in pixels.

**viewportSelection**

viewportSelection: **Selection** <any, any, any, any>

A d3 selection of the Chart's viewport.

**viewportWidthPx**

viewportWidthPx: **number**

The stored width of the viewport SVG in pixels.

**xScale**

xScale: **ScaleLinear** <number, number>

A D3 scale that the Chart will use to translate between semantic and viewport coordinates. This scale will be periodically re-scaled after zoom events.

**yScale**

yScale: (row: **number**): **number**

A simple function that maps from row numbers to the pixel y value of the corresponding row.

**zoomConstraint**

zoomConstraint: **None**

A Chart's contents are scaled by a scaling factor k. If a zoomConstraint of the form [min\_k, max\_k] is provided, the scaling factor will be constrained to that range. This will not constrain panning.

**zoomable**

zoomable: **boolean**

This controls whether or not the Chart has zooming enabled.

## Accessors

### divHeight

```
get divHeight(): undefined | string | number
```

Gets the divHeight property.

```
set divHeight(value: undefined | string | number): void
```

Sets the divHeight property. This directly adjusts the height CSS style property on the Chart's div element.

### divMargin

```
get divMargin(): undefined | string | number
```

Gets the divMargin property.

```
set divMargin(value: undefined | string | number): void
```

Sets the divMargin property. This directly adjusts the margin CSS style property on the Chart's div element.

### divOutline

```
get divOutline(): undefined | string
```

Gets the divOutline property.

```
set divOutline(value: undefined | string): void
```

Sets the divOutline property. This directly adjusts the outline CSS style property on the Chart's div element.

### divOverflowX

```
get divOverflowX(): undefined | string
```

Gets the divOverflowX property.

```
set divOverflowX(value: undefined | string): void
```

Sets the divOverflowX property. This directly adjusts the overflow-x CSS style property on the Chart's div element.

### **divOverflowY**

```
get divOverflowY(): undefined | string
```

Gets the divOverflowY property.

```
set divOverflowY(value: undefined | string): void
```

Sets the divOverflowY property. This directly adjusts the overflow-y CSS style property on the Chart's div element.

### **divWidth**

```
get divWidth(): undefined | string | number
```

Gets the divWidth property.

```
set divWidth(value: undefined | string | number): void
```

Sets the divWidth property. This directly adjusts the width CSS style property on the Chart's div element.

### **domain**

```
get domain(): None
```

Gets the domain of the Chart's x scale.

```
set domain(domain: None): void
```

Set the domain of the Chart's x scale.

### **range**

```
get range(): None
```

Gets the range of the Chart's x scale.

```
set range(range: None): void
```

Set the range of the Chart's x scale.

### **renderParams**

```
get renderParams(): P
```

Getter for the Chart's most recently used RenderParams.

```
set renderParams(params: P): void
```

Setter for the renderParams property.

## transform

```
get transform(): Transform
```

Getter for the transform property. This also updates the internal transform on the Chart's pad DOM element.

```
set transform(transform: Transform): void
```

Setter for the transform property.

## viewportHeight

```
get viewportHeight(): undefined | string | number
```

Gets the viewportHeight property.

```
set viewportHeight(value: undefined | string | number): void
```

Sets the viewportHeight property. This directly adjusts the height SVG attribute on the Chart's viewport SVG element.

## viewportWidth

```
get viewportWidth(): undefined | string | number
```

Gets the viewportWidth property.

```
set viewportWidth(value: undefined | string | number): void
```

Sets the viewportWidth property. This directly adjusts the width SVG attribute on the Chart's viewport SVG element.

## Methods

### addAxis

```
addAxis(): void
```

This adds a horizontal axis glyph to the top of the Chart.

**Returns:** void

### addGlyphModifier

```
addGlyphModifier(modifier: GlyphModifier <A, C>, initialize: boolean): void
```

This adds a GlyphModifier to the Chart.

#### Type parameters

- A: Annotation
- C: Chart

### Parameters

- modifier: GlyphModifier <A, C>
- initialize: boolean

**Returns:** void

### addRowStripes

```
addRowStripes(): void
```

If the rowColors property has been defined, this method adds row stripes to the Chart.

**Returns:** void

### alertObservers

```
alertObservers(): void
```

This calls each of this Chart's attached observer's alert() method.

**Returns:** void

### applyGlyphModifiers

```
applyGlyphModifiers(): void
```

This applies each of the Chart's GlyphModifier.zoom() methods, resulting in each of the glyphs in the Chart being appropriately redrawn for the current zoom level.

**Returns:** void

### calculateContainerDimensions

```
calculateContainerDimensions(): DOMRect
```

This returns a DOMRect that describes the bounding box of the Chart's container.

**Returns:** DOMRect

### calculateContainerHeight

```
calculateContainerHeight(): number
```

This calculates and returns the Chart's DOM container's height in pixels.

**Returns:** number

### calculateContainerWidth

```
calculateContainerWidth(): number
```

This calculates and returns the Chart's DOM container's width in pixels.

**Returns:** number

### calculateDivDimensions

```
calculateDivDimensions(): DOMRect
```

This returns a DOMRect that describes the bounding box of the Chart's div.

**Returns:** DOMRect

### calculatePadDimensions

```
calculatePadDimensions(): DOMRect
```

This returns a DOMRect that describes the SVG pad dimensions.

**Returns:** DOMRect

### calculatePadHeight

```
calculatePadHeight(): number
```

This calculates and returns the height of the SVG pad in pixels.

**Returns:** number

### calculatePadWidth

```
calculatePadWidth(): number
```

This calculates and returns the width of the SVG pad in pixels.

**Returns:** number

### calculateViewportDimensions

```
calculateViewportDimensions(): DOMRect
```

This returns a DOMRect that describes the bounding box of the viewport.

**Returns:** DOMRect

**calculateViewportHeight**

```
calculateViewportHeight(): number
```

This calculates and returns the height of the SVG viewport in pixels.

**Returns:** number

**calculateViewportWidth**

```
calculateViewportWidth(): number
```

This calculates and returns the width of the SVG viewport in pixels.

**Returns:** number

**clear**

```
clear(): void
```

This method clears all glyphs that have been rendered in the Chart.

**Returns:** void

**clearHighlight**

```
clearHighlight(selector: string): void
```

Clear highlights from the Chart. If a selector is supplied, only the highlight that matches that selector will be removed. Otherwise, all highlights will be removed.

**Parameters**

- selector: string

**Returns:** void

**configureResize**

```
configureResize(): void
```

This configures the Chart to respond to browser resize events. The default resize behavior is for the Chart to maintain the current semantic view range, either stretching or shrinking the current view.

**Returns:** void



### configureZoom

```
configureZoom(): void
```

This configures the chart's viewport to appropriately handle browser zoom events.

**Returns:** void

### defaultDraw

```
defaultDraw(params: P): void
```

The default draw() callback. It adds a horizontal axis and renders the RenderParams.annotations property as rectangles.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

### defaultPostRender

```
defaultPostRender(): void
```

The default postRender() callback. It calls the Chart.applyGlyphModifiers() method.

#### Type parameters

- P: RenderParams

**Returns:** void

### defaultUpdateDimensions

```
defaultUpdateDimensions(params: P): void
```

The default updateDimensions() callback. It calls updateDivHeight(), updatePadHeight(), and updateViewportHeight(). The result is that the Chart should be tall enough to render the number of rows specified in the rowCount property.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

### defaultUpdateDomain

```
defaultUpdateDomain(params: P): void
```

The default updateDomain() callback. If the start and end properties are set on the RenderParams, it uses those to set the domain. If they are not defined, it finds the minimum start and maximum end amongst the annotations property on the RenderParams. If there are no annotations on the RenderParams, it leaves the domain alone.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

### defaultUpdateLayout

```
defaultUpdateLayout(params: P): void
```

The default updateLayout() callback. It calls intervalGraphLayout() on the annotations property of the provided RenderParams.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

### defaultUpdateRowCount

```
defaultUpdateRowCount(params: P): void
```

The default updateRowCount() callback. It sets Chart.rowCount equal to Chart.layout.rowCount.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

## disableZoom

```
disableZoom(): void
```

This disables zooming on the Chart.

**Returns:** void

## domainFromMouseEvent

```
domainFromMouseEvent(transform: Transform, sourceEvent: WheelEvent, ↵  
↵ domainConstraint: None): None
```

This method produces a new domain from a browser mousemove event.

### Parameters

- transform: Transform
- sourceEvent: WheelEvent
- domainConstraint: None

**Returns:** None

## domainFromWheelEvent

```
domainFromWheelEvent(transform: Transform, sourceEvent: WheelEvent, domainConstraint: ↵  
↵ None): None
```

This method produces a new domain from a browser wheel event.

### Parameters

- transform: Transform
- sourceEvent: WheelEvent
- domainConstraint: None

**Returns:** None

## highlight

```
highlight(config: HighlightConfig): string
```

This method highlights a region in the Chart. If no selector is provided, one will be auto generated and returned by the function.

### Parameters

- config: HighlightConfig

**Returns:** string

### **initializeXScale**

```
initializeXScale(start: number, end: number): void
```

This initializes an x translation scale with the provided coordinates and the dimensions of the Chart.

#### **Parameters**

- start: number
- end: number

**Returns:** void

### **removeRowStripes**

```
removeRowStripes(): void
```

If they have been added, this method removes row stripes from the Chart.

**Returns:** void

### **render**

```
render(params: P): void
```

This method executes the default rendering routine. It executes each rendering callback function in succession.

#### **Parameters**

- params: P

**Returns:** void

### **resetTransform**

```
resetTransform(): void
```

Reset the Chart's transform to the zoom identity (no translation, no zoom).

**Returns:** void

### **resize**

```
resize(): void
```

This resizes the Chart. If the Chart has resizing enabled, this is called automatically when a browser zoom event occurs.

**Returns:** void

### setDivStyle

```
setDivStyle(property: string, value: undefined | string): void
```

Sets a style property on the Chart's div selection.

#### Parameters

- property: string
- value: undefined | string

**Returns:** void

### setPadAttribute

```
setPadAttribute(attribute: string, value: undefined | string): void
```

Sets an attribute on the Chart's SVG pad.

#### Parameters

- attribute: string
- value: undefined | string

**Returns:** void

### setViewportAttribute

```
setViewportAttribute(attribute: string, value: undefined | string): void
```

Sets an attribute on the Chart's SVG viewports.

#### Parameters

- attribute: string
- value: undefined | string

**Returns:** void

### updateDivHeight

```
updateDivHeight(): void
```

This updates the Chart's div height to accommodate the rowHeight, rowCount, and pad sizes. If Chart.\_divHeight is explicitly defined, this will do nothing.

**Returns:** void

**updateDivWidth**

```
updateDivWidth(): void
```

This sets the Chart's div width to 100%. If Chart.\_divWidth is explicitly defined, this will do nothing.

**Returns:** void

**updatePadHeight**

```
updatePadHeight(): void
```

This updates the Chart's SVG pad height to accommodate the rowHeight, rowCount, and the upper/lower pad sizes. If Chart.\_padHeight is explicitly defined, this will do nothing.

**Returns:** void

**updateRange**

```
updateRange(): void
```

This sets the Chart.xScale range to [0, viewportWidthPx]

**Returns:** void

**updateViewportHeight**

```
updateViewportHeight(): void
```

This updates the Chart's SVG viewport heights to accommodate the rowHeight and rowCount. If Chart.\_viewportHeight is explicitly defined, this will do nothing.

**Returns:** void

**updateViewportPosition**

```
updateViewportPosition(): void
```

This updates the Chart's SVG viewport positions to accommodate the left and upper pad sizes.

**Returns:** void

### updateViewportProperties

```
updateViewportProperties(): void
```

This updates all of the viewport properties by calling `updateViewportHeight()`, `updateViewportWidth()`, and `updateViewportPosition()`.

**Returns:** void

### updateViewportWidth

```
updateViewportWidth(): void
```

This updates the Chart's SVG viewport width to accommodate the left and right pad sizes. If `Chart._viewportWidth` is explicitly defined, this will do nothing.

**Returns:** void

### zoom

```
zoom(): void
```

This is the handler method that will be called when the Chart's viewport receives a browser zoom event.

**Returns:** void

### zoomHighlight

```
zoomHighlight(): void
```

This zooms the Chart highlights.

**Returns:** void

### getDomainFromAnnotations

```
getDomainFromAnnotations(annotations: Annotation []): None
```

Returns a domain given a list of Annotations.

#### Type parameters

- P: RenderParams

#### Parameters

- annotations: Annotation []

**Returns:** None

### 3.1.3 ChartObserver

```
class ChartObserver
```

An abstract class for objects that “observe” Charts.

#### Constructors

```
() : ChartObserver
```

#### Properties

##### charts

```
charts: Chart <any> []
```

A list of Charts that the Plugin will pay attention to.

#### Methods

##### add

```
add(chart: Chart | Chart <any> []): void
```

This method registers a Chart or list of Charts with the Plugin.

##### Parameters

- chart: Chart | Chart <any> []

**Returns:** void

##### addChart

```
addChart(chart: Chart <any>): void
```

Add a Chart to the observer.

##### Parameters

- chart: Chart <any>

**Returns:** void



## alert

```
alert(chart: Chart <any>): void
```

The method that will be called when the observer is alerted by a Chart.

### Parameters

- chart: Chart <any>

**Returns:** void

## 3.1.4 RadialChart

```
class RadialChart<P extends RenderParams>
```

This Chart class is designed for rendering features in a circular context, e.g. bacterial genomes.

### Type parameters

- P: RenderParams

### Constructors

```
(config: RadialChartConfig <P>): RadialChart
```

### Type parameters

- P: RenderParams

### Parameters

- config: RadialChartConfig

### Properties

#### \_divHeight

```
_divHeight: undefined | string
```

The CSS height property of the Chart's div.

#### \_divMargin

```
_divMargin: undefined | string | number
```

The CSS margin property of the Chart's div.

### **\_divOutline**

`_divOutline: undefined | string`

The CSS outline property of the Chart's div.

### **\_divOverflowX**

`_divOverflowX: undefined | string`

The CSS overflow-x property of the Chart's div.

### **\_divOverflowY**

`_divOverflowY: undefined | string`

The CSS overflow-y property of the Chart's div.

### **\_divWidth**

`_divWidth: undefined | string`

The CSS width property of the Chart's div.

### **\_padHeight**

`_padHeight: undefined | string`

The stored value of the pad SVG height property.

### **\_padWidth**

`_padWidth: undefined | string`

The stored value of the pad SVG width property.

### **\_renderParams**

`_renderParams: undefined | P`

The last used render parameters.

**\_transform**

```
_transform: Transform
```

The Transform object that describes the current zoom transformation.

**\_viewportHeight**

```
_viewportHeight: undefined | string
```

The stored value of the viewport SVG height property.

**\_viewportWidth**

```
_viewportWidth: undefined | string
```

The stored value of the viewport SVG width property.

**axisRadius**

```
axisRadius: undefined | number
```

The radius of the circle that defines the axis placement.

**containerSelection**

```
containerSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's DOM container. This is a pre-existing DOM element (probably a div).

**defSelection**

```
defSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's defs element. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Element/defs>

**divSelection**

```
divSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's div container. This is created when the Chart is instantiated and placed inside of the selected container in the DOM.

**domainConstraint**

```
domainConstraint: (chart: Chart <P>): None
```

This constrains the Chart's domain, which in turn constrains both zoom level and panning. The parameter is a callback function that is evaluated after each zoom event to produce an interval that constrains the domain.

**draw**

```
draw: (params: P): void
```

The rendering callback that should be responsible for drawing glyphs with the rendering API.

**glyphModifiers**

```
glyphModifiers: GlyphModifier <any, any> []
```

A list of GlyphModifiers that control the glyphs rendered in the Chart.

**highlightSelection**

```
highlightSelection: Selection <any, any, any, any>
```

A d3 selection of the Chart's highlight.

**id**

```
id: string
```

A unique identifier for the Chart.

**initialDomain**

```
initialDomain: None
```

The initialized domain of the Chart when render() is called with the initializeXScale flag.

### innerRadius

innerRadius: **number**

The inner radius of the conceptual annulus that defines the Chart annotation track.

### layout

layout: **VerticalLayout**

### leftPadSize

leftPadSize: **number**

The number of pixels of padding on the left side of the Chart.

### lowerPadSize

lowerPadSize: **number**

The number of pixels of padding on the bottom of the Chart.

### observers

observers: **ChartObserver** []

A list of observers attached to the Chart.

### outerRadius

outerRadius: **number**

The outer radius of the conceptual annulus that defines the Chart annotation track.

### overflowViewportSelection

overflowViewportSelection: **Selection** <any, any, any, any>

A d3 selection of the Chart's viewport that allows rendering overflow.

**padSelection**

padSelection: **Selection** <any, any, any, any>

A d3 selection of the viewport's padding container.

**padSize**

padSize: **number**

The number of pixels of padding around each edge of the Chart.

**postRender**

postRender: (params: **P**): **void**

The callback function that the Chart executes after render() is called.

**postResize**

postResize: (): **void**

The callback function that the Chart executes after resize() is called.

**postZoom**

postZoom: (): **void**

The callback function that the Chart executes after zoom() is called.

**resizable**

resizable: **boolean**

This controls whether or not the Chart has automatic resizing enabled.

**rightPadSize**

rightPadSize: **number**

The number of pixels of padding on the right side of the Chart.

### rowColors

```
rowColors: undefined | string []
```

A list of colors that will color the Chart's rows in a repeating pattern.

### rowCount

```
rowCount: number
```

The number of rows in the Chart.

### rowHeight

```
rowHeight: number
```

The height in pixels of a horizontal row in the Chart. This defaults to a value of 10.

### rowOpacity

```
rowOpacity: number
```

The opacity of the colored row stripes.

### selector

```
selector: string
```

A string that can be used to uniquely select the target DOM container.

### trackHeight

```
trackHeight: number
```

The "height" of the radial track on which annotations will be rendered. Conceptually, this is equal to the difference of the radii of two concentric circles that define an annulus.

### trackOutlineSelection

```
trackOutlineSelection: undefined | Selection <any, any, any, any>
```

A d3 selection to the track outline.

**updateDimensions**

updateDimensions: (params: **P**): **void**

The rendering callback function that should be responsible for updating the Chart's DOM element dimensions.

**updateDomain**

updateDomain: (params: **P**): **void**

The rendering callback function that should be responsible for updating the domain of the Chart.xScale property.

**updateLayout**

updateLayout: (params: **P**): **void**

The rendering callback function that should be responsible for updating the Chart.layout property.

**updateRowCount**

updateRowCount: (params: **P**): **void**

The rendering callback function that should be responsible for updating the Chart.rowCount property.

**upperPadSize**

upperPadSize: **number**

The number of pixels of padding on the top of the Chart.

**viewportHeightPx**

viewportHeightPx: **number**

The stored height of the viewport SVG in pixels.

**viewportSelection**

viewportSelection: **Selection** <any, any, any, any>

A d3 selection of the Chart's viewport.



### viewportWidthPx

```
viewportWidthPx: number
```

The stored width of the viewport SVG in pixels.

### xScale

```
xScale: ScaleLinear <number, number>
```

A D3 scale that the Chart will use to translate between semantic and viewport coordinates. This scale will be periodically re-scaled after zoom events.

### yScale

```
yScale: (row: number): number
```

A simple function that maps from row numbers to the pixel y value of the corresponding row.

### zoomConstraint

```
zoomConstraint: None
```

A Chart's contents are scaled by a scaling factor  $k$ . If a zoomConstraint of the form  $[\text{min\_k}, \text{max\_k}]$  is provided, the scaling factor will be constrained to that range. This will not constrain panning.

### zoomable

```
zoomable: boolean
```

This controls whether or not the Chart has zooming enabled.

## Accessors

### divHeight

```
get divHeight(): undefined | string | number
```

Gets the divHeight property.

```
set divHeight(value: undefined | string | number): void
```

Sets the divHeight property. This directly adjusts the height CSS style property on the Chart's div element.

### **divMargin**

```
get divMargin(): undefined | string | number
```

Gets the divMargin property.

```
set divMargin(value: undefined | string | number): void
```

Sets the divMargin property. This directly adjusts the margin CSS style property on the Chart's div element.

### **divOutline**

```
get divOutline(): undefined | string
```

Gets the divOutline property.

```
set divOutline(value: undefined | string): void
```

Sets the divOutline property. This directly adjusts the outline CSS style property on the Chart's div element.

### **divOverflowX**

```
get divOverflowX(): undefined | string
```

Gets the divOverflowX property.

```
set divOverflowX(value: undefined | string): void
```

Sets the divOverflowX property. This directly adjusts the overflow-x CSS style property on the Chart's div element.

### **divOverflowY**

```
get divOverflowY(): undefined | string
```

Gets the divOverflowY property.

```
set divOverflowY(value: undefined | string): void
```

Sets the divOverflowY property. This directly adjusts the overflow-y CSS style property on the Chart's div element.

### **divWidth**

```
get divWidth(): undefined | string | number
```

Gets the divWidth property.

```
set divWidth(value: undefined | string | number): void
```

Sets the divWidth property. This directly adjusts the width CSS style property on the Chart's div element.

## domain

```
get domain(): None
```

Gets the domain of the Chart's x scale.

```
set domain(domain: None): void
```

Set the domain of the Chart's x scale.

## range

```
get range(): None
```

Gets the range of the Chart's x scale.

```
set range(range: None): void
```

Set the range of the Chart's x scale.

## renderParams

```
get renderParams(): P
```

Getter for the Chart's most recently used RenderParams.

```
set renderParams(params: P): void
```

Setter for the renderParams property.

## transform

```
get transform(): Transform
```

Getter for the transform property. This also updates the internal transform on the Chart's pad DOM element.

```
set transform(transform: Transform): void
```

Setter for the transform property.

## viewportHeight

```
get viewportHeight(): undefined | string | number
```

Gets the viewportHeight property.

```
set viewportHeight(value: undefined | string | number): void
```

Sets the viewportHeight property. This directly adjusts the height SVG attribute on the Chart's viewport SVG element.

## viewportWidth

```
get viewportWidth(): undefined | string | number
```

Gets the viewportWidth property.

```
set viewportWidth(value: undefined | string | number): void
```

Sets the viewportWidth property. This directly adjusts the width SVG attribute on the Chart's viewport SVG element.

## Methods

### addAxis

```
addAxis(): void
```

**Returns:** void

### addGlyphModifier

```
addGlyphModifier(modifier: GlyphModifier <A, C>, initialize: boolean): void
```

This adds a GlyphModifier to the Chart.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- modifier: GlyphModifier <A, C>
- initialize: boolean

**Returns:** void

### addRowStripes

```
addRowStripes(): void
```

If the rowColors property has been defined, this method adds row stripes to the Chart.

**Returns:** void

### addTrackOutline

```
addTrackOutline(): void
```

**Returns:** void

### alertObservers

```
alertObservers(): void
```

This calls each of this Chart's attached observer's alert() method.

**Returns:** void

### applyGlyphModifiers

```
applyGlyphModifiers(): void
```

This applies each of the Chart's GlyphModifier.zoom() methods, resulting in each of the glyphs in the Chart being appropriately redrawn for the current zoom level.

**Returns:** void

### calculateContainerDimensions

```
calculateContainerDimensions(): DOMRect
```

This returns a DOMRect that describes the bounding box of the Chart's container.

**Returns:** DOMRect

### calculateContainerHeight

```
calculateContainerHeight(): number
```

This calculates and returns the Chart's DOM container's height in pixels.

**Returns:** number

### calculateContainerWidth

```
calculateContainerWidth(): number
```

This calculates and returns the Chart's DOM container's width in pixels.

**Returns:** number

### calculateDivDimensions

```
calculateDivDimensions(): DOMRect
```

This returns a DOMRect that describes the bounding box of the Chart's div.

**Returns:** DOMRect

### calculatePadDimensions

```
calculatePadDimensions(): DOMRect
```

This returns a DOMRect that describes the SVG pad dimensions.

**Returns:** DOMRect

### calculatePadHeight

```
calculatePadHeight(): number
```

This calculates and returns the height of the SVG pad in pixels.

**Returns:** number

### calculatePadWidth

```
calculatePadWidth(): number
```

This calculates and returns the width of the SVG pad in pixels.

**Returns:** number

### calculateViewportDimensions

```
calculateViewportDimensions(): DOMRect
```

This returns a DOMRect that describes the bounding box of the viewport.

**Returns:** DOMRect

### calculateViewportHeight

```
calculateViewportHeight(): number
```

This calculates and returns the height of the SVG viewport in pixels.

**Returns:** number

### calculateViewportWidth

```
calculateViewportWidth(): number
```

This calculates and returns the width of the SVG viewport in pixels.

**Returns:** number

### clear

```
clear(): void
```

This method clears all glyphs that have been rendered in the Chart.

**Returns:** void

### clearHighlight

```
clearHighlight(selector: string): void
```

#### Parameters

- selector: string

**Returns:** void

### configureResize

```
configureResize(): void
```

This configures the Chart to respond to browser resize events. The default resize behavior is for the Chart to maintain the current semantic view range, either stretching or shrinking the current view.

**Returns:** void

### configureZoom

```
configureZoom(): void
```

**Returns:** void

### defaultDraw

```
defaultDraw(params: P): void
```

The default draw() callback. It adds a horizontal axis and renders the RenderParams.annotations property as rectangles.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

### defaultPostRender

```
defaultPostRender(): void
```

The default postRender() callback. It calls the Chart.applyGlyphModifiers() method.

#### Type parameters

- P: RenderParams

**Returns:** void

### defaultUpdateDimensions

```
defaultUpdateDimensions(params: P): void
```

The default updateDimensions() callback. It calls updateDivHeight(), updatePadHeight(), and updateViewportHeight(). The result is that the Chart should be tall enough to render the number of rows specified in the rowCount property.

#### Type parameters

- P: RenderParams

#### Parameters

- params: P

**Returns:** void

### defaultUpdateDomain

```
defaultUpdateDomain(params: P): void
```

The default updateDomain() callback. If the start and end properties are set on the RenderParams, it uses those to set the domain. If they are not defined, it finds the minimum start and maximum end amongst the annotations property on the RenderParams. If there are no annotations on the RenderParams, it leaves the domain alone.

#### Type parameters

- P: RenderParams



**Parameters**

- params: P

**Returns:** void**defaultUpdateLayout**

```
defaultUpdateLayout(params: P): void
```

The default updateLayout() callback. It calls intervalGraphLayout() on the annotations property of the provided RenderParams.

**Type parameters**

- P: RenderParams

**Parameters**

- params: P

**Returns:** void**defaultUpdateRowCount**

```
defaultUpdateRowCount(params: P): void
```

The default updateRowCount() callback. It sets Chart.rowCount equal to Chart.layout.rowCount.

**Type parameters**

- P: RenderParams

**Parameters**

- params: P

**Returns:** void**disableZoom**

```
disableZoom(): void
```

This disables zooming on the Chart.

**Returns:** void

### domainFromMouseEvent

```
domainFromMouseEvent(transform: Transform, sourceEvent: WheelEvent): None
```

#### Parameters

- transform: Transform
- sourceEvent: WheelEvent

**Returns:** None

### domainFromWheelEvent

```
domainFromWheelEvent(transform: Transform, sourceEvent: WheelEvent): None
```

#### Parameters

- transform: Transform
- sourceEvent: WheelEvent

**Returns:** None

### fitRadialDimensions

```
fitRadialDimensions(): void
```

**Returns:** void

### highlight

```
highlight(config: HighlightConfig): string
```

#### Parameters

- config: HighlightConfig

**Returns:** string

### initializeXScale

```
initializeXScale(start: number, end: number): void
```

This initializes an x translation scale with the provided coordinates and the dimensions of the Chart.

#### Parameters

- start: number
- end: number

**Returns:** void

### removeRowStripes

```
removeRowStripes(): void
```

If they have been added, this method removes row stripes from the Chart.

**Returns:** void

### render

```
render(params: P): void
```

This method executes the default rendering routine. It executes each rendering callback function in succession.

#### Parameters

- params: P

**Returns:** void

### renderAxis

```
renderAxis(): void
```

**Returns:** void

### renderTrackOutline

```
renderTrackOutline(): void
```

**Returns:** void

### resetTransform

```
resetTransform(): void
```

Reset the Chart's transform to the zoom identity (no translation, no zoom).

**Returns:** void

### resize

```
resize(): void
```

**Returns:** void

### **setDivStyle**

```
setDivStyle(property: string, value: undefined | string): void
```

Sets a style property on the Chart's div selection.

#### **Parameters**

- property: string
- value: undefined | string

**Returns:** void

### **setPadAttribute**

```
setPadAttribute(attribute: string, value: undefined | string): void
```

Sets an attribute on the Chart's SVG pad.

#### **Parameters**

- attribute: string
- value: undefined | string

**Returns:** void

### **setViewportAttribute**

```
setViewportAttribute(attribute: string, value: undefined | string): void
```

Sets an attribute on the Chart's SVG viewports.

#### **Parameters**

- attribute: string
- value: undefined | string

**Returns:** void

### **squareToDivWidth**

```
squareToDivWidth(): void
```

**Returns:** void

### updateDivHeight

```
updateDivHeight(): void
```

This updates the Chart's div height to accommodate the rowHeight, rowCount, and pad sizes. If Chart.\_divHeight is explicitly defined, this will do nothing.

**Returns:** void

### updateDivWidth

```
updateDivWidth(): void
```

This sets the Chart's div width to 100%. If Chart.\_divWidth is explicitly defined, this will do nothing.

**Returns:** void

### updatePadHeight

```
updatePadHeight(): void
```

This updates the Chart's SVG pad height to accommodate the rowHeight, rowCount, and the upper/lower pad sizes. If Chart.\_padHeight is explicitly defined, this will do nothing.

**Returns:** void

### updateRange

```
updateRange(): void
```

**Returns:** void

### updateViewportHeight

```
updateViewportHeight(): void
```

**Returns:** void

### updateViewportPosition

```
updateViewportPosition(): void
```

This updates the Chart's SVG viewport positions to accommodate the left and upper pad sizes.

**Returns:** void

**updateViewportProperties**

`updateViewportProperties(): void`

This updates all of the viewport properties by calling `updateViewportHeight()`, `updateViewportWidth()`, and `updateViewportPosition()`.

**Returns:** void

**updateViewportWidth**

`updateViewportWidth(): void`

**Returns:** void

**zoom**

`zoom(): void`

**Returns:** void

**zoomHighlight**

`zoomHighlight(): void`

**Returns:** void

**getDomainFromAnnotations**

`getDomainFromAnnotations(annotations: Annotation []): None`

Returns a domain given a list of Annotations.

**Type parameters**

- P: RenderParams

**Parameters**

- annotations: Annotation []

**Returns:** None

### 3.1.5 ZoomSyncer

```
class ZoomSyncer
```

This class can be used to synchronize the zoom level across different Charts.

#### Constructors

```
() : ZoomSyncer
```

#### Properties

##### charts

```
charts: Chart <any> []
```

A list of Charts that the Plugin will pay attention to.

#### Methods

##### add

```
add(chart: Chart | Chart <any> []): void
```

This method registers a Chart or list of Charts with the Plugin.

##### Parameters

- chart: Chart | Chart <any> []

**Returns:** void

##### addChart

```
addChart(chart: Chart <any>): void
```

Add a Chart to the observer.

##### Parameters

- chart: Chart <any>

**Returns:** void

## alert

```
alert(caller: Chart <any>): void
```

The ZoomZyncer alert method synchronizes all of the Transforms on each of the Charts it is observing and fires the zooming functionality.

### Parameters

- caller: Chart <any>

**Returns:** void

## 3.2 Interfaces

### 3.2.1 AggregationConfig

```
interface AggregationConfig<A extends Annotation>
```

This defines the parameters for a call to an Annotation aggregation function.

### Type parameters

- A: Annotation

### Properties

#### annotations

```
annotations: A []
```

The list of Annotations to be aggregated.

#### criterion

```
criterion: (a: A, b: A): boolean
```

The comparison function that serves as the criterion for aggregation.

#### idPrefix

```
idPrefix: undefined | string
```

The ID prefix for each resulting AnnotationGroup. E.g. if the idPrefix “group” is supplied, the resulting groups will have IDs of the form: “group-1,” “group-2,” etc.



### 3.2.2 AlignmentAnnotations

```
interface AlignmentAnnotations
```

The return type for the `getAlignmentAnnotations()` function.

#### Properties

##### gaps

```
gaps: SequenceAnnotation
```

##### insertions

```
insertions: SequenceAnnotation []
```

##### matches

```
matches: SequenceAnnotation
```

##### substitutions

```
substitutions: SequenceAnnotation
```

### 3.2.3 AlignmentConfig

```
interface AlignmentConfig
```

This defines the parameters for a call to the `getAlignmentAnnotations()` function.

#### Properties

##### end

```
end: undefined | number
```

**id**

```
id: string
```

**query**

```
query: string
```

**start**

```
start: number
```

**target**

```
target: string
```

### 3.2.4 Annotation

```
interface Annotation
```

**Properties**

**end**

```
end: number
```

The end coordinate of the Annotation.

**id**

```
id: string
```

A unique identifier for the Annotation.

**start**

```
start: number
```

The start coordinate of the Annotation.

**3.2.5 AnnotationDatum**

```
interface AnnotationDatum<A extends Annotation, C extends Chart>
```

An interface that simply joins an Annotation object and a Chart is has been rendered in.

**Type parameters**

- A: Annotation
- C: Chart

**Properties****a**

```
a: A
```

The Annotation object.

**c**

```
c: C
```

The Chart object.

**3.2.6 AnnotationGenerationConfig**

```
interface AnnotationGenerationConfig
```

An interface that defines the parameters for a call to the generateAnnotations function.

**Properties****generationPattern**

```
generationPattern: undefined | Sequential | Random
```

### maxX

maxX: **undefined** | **number**

### maxY

maxY: **undefined** | **number**

### n

n: **number**

### pad

pad: **undefined** | **number**

### startY

startY: **undefined** | **number**

### width

width: **undefined** | **number**

## 3.2.7 AnnotationGroupConfig

```
interface AnnotationGroupConfig<A extends Annotation>
```

### Type parameters

- A: Annotation

### Properties

#### annotations

annotations: **undefined** | A []

**end**`end: undefined | number`**id**`id: undefined | string`**start**`start: undefined | number`

### 3.2.8 ArcConfig

`interface ArcConfig<A extends Annotation, C extends Chart>`

An interface that defines the parameters for a call to the arc rendering function.

#### Type parameters

- A: Annotation
- C: Chart

#### Properties

##### annotations

`annotations: A []`

A list of Annotation objects that will be used to render the glyphs.

##### chart

`chart: C`

The Chart object in which the glyphs will be rendered.

**fillColor**

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

**fillOpacity**

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

**height**

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

**initializeFn**

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

**row**

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

**selector**

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

### strokeColor

strokeColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the color of the border around the glyph.

### strokeDashArray

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

strokeDashOffset: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

strokeLineJoin: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

### strokeWidth

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

### target

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

### x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

### y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

### zoomFn

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.



### 3.2.9 AreaConfig

```
interface AreaConfig<A extends PlotAnnotation, C extends Chart>
```

An interface that defines the parameters for a call to the area rendering function.

#### Type parameters

- A: PlotAnnotation
- C: Chart

#### Properties

##### annotations

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

##### chart

```
chart: C
```

The Chart object in which the glyphs will be rendered.

##### domain

```
domain: undefined | None | GlyphCallback <A, C, None>
```

This defines the domain of the plot.

##### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

##### fillDirection

```
fillDirection: undefined | Up | Down | GlyphCallback <A, C, FillDirection>
```

**fillOpacity**

fillOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the fill opacity of the glyph.

**height**

height: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel height of the glyph.

**initializeFn**

initializeFn: **undefined** | **()**: **void**

**pathData**

pathData: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback that returns a string that defines the line's SVG path

**range**

range: **undefined** | **None** | GlyphCallback <A, C, **None**>

This defines the range of the plot.

**row**

row: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the row that the glyph is placed in.

**rowSpan**

rowSpan: **undefined** | **number**

The number of bins that the plot will span. This defaults to 1, which forces the plot to fit into one row. If an argument is supplied, it will cause the plot to grow downward. It will have no effect if a custom lineFunc is supplied.

**selector**

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

**strokeColor**

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

**strokeDashArray**

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

**strokeLineJoin**

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

### strokeWidth

strokeWidth: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the width of the border around the glyph.

### target

target: **undefined** | Selection <any, any, any, any> | Viewport | Overflow | Defs

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

width: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel width of the glyph.

### x

x: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel x coordinate of the glyph.

### y

y: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel y coordinate of the glyph

## zoomFn

```
zoomFn: undefined | (): void
```

### 3.2.10 AugmentConfig

```
interface AugmentConfig<T extends generic>
```

An interface that defines the parameters for a call to the augment function.

#### Type parameters

- T: generic

#### Properties

##### end

```
end: undefined | AugmentParam <T, number>
```

An AugmentParam that describes how to compute the Annotation.end property.

##### id

```
id: undefined | AugmentParam <T, string>
```

An AugmentParam that describes how to compute the Annotation.id property.

#### objects

```
objects: T []
```

A list of type T, i.e. an arbitrary object representation of annotation records that fail to implement Annotation.

#### skipValidate

```
skipValidate: undefined | boolean
```

If this is set to true, this skips the validation on the returned objects. That means that the function will be happy to return objects that fail to implement Annotation.

**start**

start: **undefined** | AugmentParam <T, **number**>

An AugmentParam that describes how to compute the Annotation.start property.

### 3.2.11 AugmentParam

**interface** AugmentParam<T **extends** generic, V **extends** generic>

An interface that describes a parameter in a call to the augment function.

**Type parameters**

- T: generic
- V: generic

**Properties****fn**

fn: (t: **T**): V

The callback function used to compute the value for the property.

**virtual**

virtual: **undefined** | **boolean**

If this is set to true, the callback function will be added as a getter on the target object. If this is false or omitted, the callback function will be evaluated once and the resulting value will be added on the object as a real property.

### 3.2.12 BarPlotConfig

**interface** BarPlotConfig<A **extends** PlotAnnotation, C **extends** Chart>

An interface that defines the parameters for a call to the barPlot rendering function.

**Type parameters**

- A: PlotAnnotation
- C: Chart

## Properties

### annotations

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

### barHeightFn

```
barHeightFn: undefined | (ann: A, value: number): number
```

### chart

```
chart: C
```

The Chart object in which the glyphs will be rendered.

### domain

```
domain: undefined | None | GlyphCallback <A, C, None>
```

This defines the domain of the plot.

### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

### fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

### height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

## initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## range

```
range: undefined | None | GlyphCallback <A, C, None>
```

This defines the range of the plot.

## row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

## rowSpan

```
rowSpan: undefined | number
```

The number of bins that the plot will span. This defaults to 1, which forces the plot to fit into one row. If an argument is supplied, it will cause the plot to grow downward. It will have no effect if a custom lineFunc is supplied.

## selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

## strokeColor

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.



### strokeDashArray

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

### strokeWidth

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

## target

target: **undefined** | Selection <any, any, any, any> | Viewport | Overflow | Defs

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

## width

width: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel width of the glyph.

## x

x: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel x coordinate of the glyph.

## y

y: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel y coordinate of the glyph

## zoomFn

zoomFn: **undefined** | **()**: **void**

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## 3.2.13 BedAnnotation

**interface** BedAnnotation

An interface that describes BED records. For more information, see <https://genome.ucsc.edu/FAQ/FAQformat.html#format1>

## Properties

### blockCount

```
blockCount: undefined | number
```

A BED field for records that should be drawn as discontinuous/fragmented glyphs. This describes the number of fragments.

### blockSizes

```
blockSizes: undefined | number []
```

A BED field for records that should be drawn as discontinuous/fragmented glyphs. This describes the size of each fragment.

### blockStarts

```
blockStarts: undefined | number []
```

A BED field for records that should be drawn as discontinuous/fragmented glyphs. This describes the offset of each fragment.

### chrom

```
chrom: string
```

A BED field that describes the chromosome of the record.

### end

```
end: number
```

A BED field that describes the ending position of the record. This is chromEnd in the BED spec, but it's end here to fit in better with the rest of SODA.

### id

```
id: string
```

A unique identifier for the Annotation.

### itemRgb

itemRgb: **undefined** | **string**

A BED field BED field that defines the color of the feature. It is an RGB string, e.g. (0, 1, 256).

### name

name: **undefined** | **string**

A BED field that describes the name of the record.

### score

score: **undefined** | **number**

A BED field that describes the “score” of the record.

### start

start: **number**

A BED field that describes the starting position of the record. This is chromStart in the BED spec, but it’s start here to fit in better with the rest of SODA.

### strand

strand: **undefined** | Forward | Reverse | Unknown | Unoriented

A BED field that describes the orientation/strand of the record.

### thickEnd

thickEnd: **undefined** | **number**

A BED field that describes at which coordinate the feature should stop being drawn “thickly.”

**thickStart**

```
thickStart: undefined | number
```

A BED field that describes at which coordinate the feature should start being drawn “thickly.”

**3.2.14 ChartConfig**

```
interface ChartConfig<P extends RenderParams>
```

This describes the parameters for configuring and initializing a Chart.

**Type parameters**

- P: RenderParams

**Properties****axisType**

```
axisType: undefined | Bottom | Top
```

This controls whether or not the Chart will render a horizontal axis.

**debugShading**

```
debugShading: undefined | boolean
```

If this is set to true, the pad and viewport will be shaded so that they are visible in the browser.

**divHeight**

```
divHeight: undefined | string | number
```

The height in pixels of the Chart’s containing div.

**divMargin**

```
divMargin: undefined | string | number
```

The CSS margin property for the Chart’s div.

**divOutline**

divOutline: **undefined** | **string**

The CSS outline property for the Chart's div.

**divOverflowX**

divOverflowX: **undefined** | **string**

The CSS overflow-x setting of the Chart's containing div.

**divOverflowY**

divOverflowY: **undefined** | **string**

The CSS overflow-y setting of the Chart's containing div.

**divWidth**

divWidth: **undefined** | **string** | **number**

The width in pixels of the Chart's containing div.

**domainConstraint**

domainConstraint: **undefined** | (chart: **Chart** <P>): **None**

This constrains the Chart's domain, which in turn constrains both zoom level and panning. The parameter is a callback function that is evaluated after each zoom event to produce an interval that constrains the domain.

**draw**

draw: **undefined** | (params: **P**): **void**

The rendering callback that should be responsible for drawing glyphs with the rendering API.

## id

```
id: undefined | string
```

A unique identifier for the Chart. This will be generated automatically if one isn't provided.

## leftPadSize

```
leftPadSize: undefined | number
```

The number of pixels of padding on the left side of the Chart.

## lowerPadSize

```
lowerPadSize: undefined | number
```

The number of pixels of padding on the bottom of the Chart.

## padSize

```
padSize: undefined | number
```

The number of pixels of padding around each edge of the Chart.

## postRender

```
postRender: undefined | (params: P): void
```

The callback function that the Chart executes after render() is called.

## postResize

```
postResize: undefined | (): void
```

The callback function that the Chart executes after resize() is called.

## postZoom

```
postZoom: undefined | (): void
```

The callback function that the Chart executes after zoom() is called.

**resizable**

resizable: **undefined** | **boolean**

This controls whether or not the Chart will automatically resize itself as it's container changes size. This will cause the Chart to ignore explicit height/width arguments in the config.

**rightPadSize**

rightPadSize: **undefined** | **number**

The number of pixels of padding on the right side of the Chart.

**rowColors**

rowColors: **undefined** | **string** []

A list of colors that will color the Chart's rows in a repeating pattern.

**rowCount**

rowCount: **undefined** | **number**

The number of rows that will be rendered.

**rowHeight**

rowHeight: **undefined** | **number**

The height in pixels of a horizontal row in the Chart. This defaults to a value of 10.

**rowOpacity**

rowOpacity: **undefined** | **number**

The opacity of the colored row stripes.



### selector

```
selector: string
```

A string that can be used to uniquely select the target DOM container.

### updateDimensions

```
updateDimensions: undefined | (params: P): void
```

The rendering callback function that should be responsible for updating the Chart's DOM element dimensions.

### updateDomain

```
updateDomain: undefined | (params: P): void
```

The rendering callback function that should be responsible for updating the domain of the Chart.xScale property.

### updateLayout

```
updateLayout: undefined | (params: P): void
```

The rendering callback function that should be responsible for updating the Chart.layout property.

### updateRowCount

```
updateRowCount: undefined | (params: P): void
```

The rendering callback function that should be responsible for updating the Chart.rowCount property.

### upperPadSize

```
upperPadSize: undefined | number
```

The number of pixels of padding on the top of the Chart.

### zoomConstraint

```
zoomConstraint: undefined | None
```

A Chart's contents are scaled by a scaling factor  $k$ . If a zoomConstraint of the form  $[\text{min\_}k, \text{max\_}k]$  is provided, the scaling factor will be constrained to that interval. This will not constrain panning.

## zoomable

zoomable: **undefined** | **boolean**

This controls whether or not the Chart will be configured to allow zooming and panning.

### 3.2.15 ChevronGlyphConfig

**interface** ChevronGlyphConfig<A **extends** Annotation, C **extends** Chart>

An interface that defines the common parameters for calls to chevron glyph rendering functions.

#### Type parameters

- A: Annotation
- C: Chart

#### Properties

##### annotations

annotations: **A** []

A list of Annotation objects that will be used to render the glyphs.

##### chart

chart: **C**

The Chart object in which the glyphs will be rendered.

##### chevronFillColor

chevronFillColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

This defines the fill color of the chevron arrows.

##### chevronFillOpacity

chevronFillOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

This defines the fill opacity of the chevron arrows.

### chevronHeight

```
chevronHeight: undefined | number | GlyphCallback <A, C, number>
```

This defines the height of the chevron arrows.

### chevronSpacing

```
chevronSpacing: undefined | number | GlyphCallback <A, C, number>
```

This defines the spacing between each chevron arrow.

### chevronStrokeColor

```
chevronStrokeColor: undefined | string | GlyphCallback <A, C, string>
```

This defines the stroke color of the chevron arrows.

### chevronStrokeOpacity

```
chevronStrokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

This defines the stroke opacity of the chevron arrows.

### chevronWidth

```
chevronWidth: undefined | number | GlyphCallback <A, C, number>
```

This defines the width of the chevron arrows.

### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

### fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

## height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

## initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## orientation

```
orientation: undefined | Forward | Reverse | Unknown | Unoriented | GlyphCallback <A, C, ↵Orientation>
```

This defines the direction that the chevron arrows will point.

## row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

## selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

## strokeColor

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

### strokeDashArray

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

strokeDashOffset: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

strokeLineJoin: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

### strokeWidth

strokeWidth: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the width of the border around the glyph.

## target

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

## width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

## x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

## y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

## zoomFn

```
zoomFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## 3.2.16 ChevronLineConfig

```
interface ChevronLineConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the chevronLine rendering function.

### Type parameters

- A: Annotation
- C: Chart

## Properties

### annotations

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

### chart

```
chart: C
```

The Chart object in which the glyphs will be rendered.

### chevronFillColor

```
chevronFillColor: undefined | string | GlyphCallback <A, C, string>
```

This defines the fill color of the chevron arrows.

### chevronFillOpacity

```
chevronFillOpacity: undefined | number | GlyphCallback <A, C, number>
```

This defines the fill opacity of the chevron arrows.

### chevronHeight

```
chevronHeight: undefined | number | GlyphCallback <A, C, number>
```

This defines the height of the chevron arrows.

### chevronSpacing

```
chevronSpacing: undefined | number | GlyphCallback <A, C, number>
```

This defines the spacing between each chevron arrow.

### **chevronStrokeColor**

```
chevronStrokeColor: undefined | string | GlyphCallback <A, C, string>
```

This defines the stroke color of the chevron arrows.

### **chevronStrokeOpacity**

```
chevronStrokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

This defines the stroke opacity of the chevron arrows.

### **chevronWidth**

```
chevronWidth: undefined | number | GlyphCallback <A, C, number>
```

This defines the width of the chevron arrows.

### **fillColor**

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

### **fillOpacity**

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

### **height**

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

### **initializeFn**

```
initializeFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.



### orientation

```
orientation: undefined | Forward | Reverse | Unknown | Unoriented | GlyphCallback <A, C, Orientation>
```

This defines the direction that the chevron arrows will point.

### row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

### selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

### strokeColor

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

### strokeDashArray

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

strokeLineJoin: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

### strokeWidth

strokeWidth: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the width of the border around the glyph.

### target

target: **undefined** | Selection <any, any, any, any> | Viewport | Overflow | Defs

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

width: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel width of the glyph.

**x****x:** **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel x coordinate of the glyph.

**y****y:** **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel y coordinate of the glyph

**zoomFn****zoomFn:** **undefined** | **()**: **void**

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

### 3.2.17 ChevronRectangleConfig

**interface** ChevronRectangleConfig<A **extends** Annotation, C **extends** Chart>

An interface that defines the parameters for a call to the chevronRectangle rendering function.

#### Type parameters

- A: Annotation
- C: Chart

#### Properties

##### annotations

**annotations:** **A** []

A list of Annotation objects that will be used to render the glyphs.

## **chart**

`chart: C`

The Chart object in which the glyphs will be rendered.

## **chevronFillColor**

`chevronFillColor: undefined | string | GlyphCallback <A, C, string>`

This defines the fill color of the chevron arrows.

## **chevronFillOpacity**

`chevronFillOpacity: undefined | number | GlyphCallback <A, C, number>`

This defines the fill opacity of the chevron arrows.

## **chevronHeight**

`chevronHeight: undefined | number | GlyphCallback <A, C, number>`

This defines the height of the chevron arrows.

## **chevronSpacing**

`chevronSpacing: undefined | number | GlyphCallback <A, C, number>`

This defines the spacing between each chevron arrow.

## **chevronStrokeColor**

`chevronStrokeColor: undefined | string | GlyphCallback <A, C, string>`

This defines the stroke color of the chevron arrows.

## **chevronStrokeOpacity**

`chevronStrokeOpacity: undefined | number | GlyphCallback <A, C, number>`

This defines the stroke opacity of the chevron arrows.

### chevronWidth

```
chevronWidth: undefined | number | GlyphCallback <A, C, number>
```

This defines the width of the chevron arrows.

### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

### fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

### height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

### initializeFn

```
initializeFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

### orientation

```
orientation: undefined | Forward | Reverse | Unknown | Unoriented | GlyphCallback <A, C,   
↪Orientation>
```

This defines the direction that the chevron arrows will point.

**row**

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

**selector**

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

**strokeColor**

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

**strokeDashArray**

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

### strokeWidth

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

### target

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

### x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

## y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

## zoomFn

```
zoomFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

### 3.2.18 ClickConfig

```
interface ClickConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the clickBehavior function.

#### Type parameters

- A: Annotation
- C: Chart

#### Properties

##### annotations

```
annotations: A []
```

The Annotations to which the interaction is applied.

##### chart

```
chart: undefined | C
```

The Chart to which the interaction is applied.



**click**

```
click: InteractionCallback <A, C>
```

A callback function that will be responsible for executing the click behavior. It will implicitly receive references to both a D3 Selection to the Annotation's representative glyph and the Annotation object itself.

**selector**

```
selector: undefined | string
```

The selector of the glyphs to which the interaction is applied.

**3.2.19 DynamicTextConfig**

```
interface DynamicTextConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the dynamicText rendering function.

**Type parameters**

- A: Annotation
- C: Chart

**Properties****alignmentBaseline**

```
alignmentBaseline: undefined | string | GlyphCallback <A, C, string>
```

How the text glyph is aligned with it's parent. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/alignment-baseline>

**annotations**

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

**chart**

chart: **C**

The Chart object in which the glyphs will be rendered.

**fillColor**

fillColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the fill color of the glyph.

**fillOpacity**

fillOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the fill opacity of the glyph.

**fontFamily**

fontFamily: **undefined** | **string** | GlyphCallback <A, C, **string**>

The font family that will be used. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-family>

**fontSize**

fontSize: **undefined** | **number** | GlyphCallback <A, C, **number**>

The font size of the text.

**fontStyle**

fontStyle: **undefined** | **string** | GlyphCallback <A, C, **string**>

The font style: normal, italic, or oblique. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-style>

### fontWeight

fontWeight: **undefined** | **string** | GlyphCallback <A, C, **string**>

The weight of the font: normal, bold, bolder, lighter. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-weight>

### height

height: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel height of the glyph.

### initializeFn

initializeFn: **undefined** | **()**: **void**

### row

row: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the row that the glyph is placed in.

### selector

selector: **undefined** | **string**

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

### strokeColor

strokeColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the color of the border around the glyph.

**strokeDashArray**

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

strokeDashOffset: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

**strokeLineJoin**

strokeLineJoin: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

**strokeOpacity**

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

**strokeWidth**

strokeWidth: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the width of the border around the glyph.

## target

target: <b>undefined</b>   Selection <any, any, any, any>   Viewport   Overflow   Defs
--

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

## text

text: <b>GlyphProperty</b> <A, C, <b>string</b> []>
---

A callback to extract a list of text to display from the represented Annotation object. It is a list of text because TextGlyphs can display varying length text depending on how much room is available at the Chart's current zoom level.

## textAnchor

textAnchor: <b>undefined</b>   <b>string</b>   GlyphCallback <A, C, <b>string</b> >
---

Where the text is aligned to: start, middle, or end. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/text-anchor>

## width

width: <b>undefined</b>   <b>number</b>   GlyphCallback <A, C, <b>number</b> >
--

A callback to define the pixel width of the glyph.

## x

x: <b>undefined</b>   <b>number</b>   GlyphCallback <A, C, <b>number</b> >
--

A callback to define the pixel x coordinate of the glyph.

## y

y: <b>undefined</b>   <b>number</b>   GlyphCallback <A, C, <b>number</b> >
--

A callback to define the pixel y coordinate of the glyph

## zoomFn

zoomFn: **undefined** | (): **void**

### 3.2.20 ExportConfig

**interface** ExportConfig<C **extends** Chart>

An interface that defines the parameters for a call to the exportPng function.

#### Type parameters

- C: Chart

#### Properties

##### chart

chart: **C**

The Chart to export.

##### filename

filename: **undefined** | **string**

The filename for the exported PNG.

##### pixelRatio

pixelRatio: **undefined** | **number**

The pixel ratio of the rendered PNG. Using a number larger than 1 will over-render the PNG, making it larger. Using smaller numbers currently has strange behavior, and it's not recommended.

### 3.2.21 FullGlyphQueryConfig

**interface** FullGlyphQueryConfig

## Properties

### annotations

```
annotations: Annotation []
```

Constrain the query to these Annotations.

### chart

```
chart: Chart <any>
```

Constrain the query to glyphs rendered in this Chart.

### selector

```
selector: string
```

Constrain the query to glyphs with this selector.

## 3.2.22 Gff3Annotation

```
interface Gff3Annotation
```

An interface that describes the fields in a GFF3 record. For more information see <http://gmod.org/wiki/GFF3/>

## Properties

### attributes

```
attributes: undefined | Map <string, string>
```

A GFF3 field that is essentially an anything goes set of key value pairs describing anything anybody every wants to add to a GFF3 record.

### end

```
end: number
```

The end coordinate of the Annotation.

**id**

id: **string**

A unique identifier for the Annotation.

**phase**

phase: **undefined** | None | None | None

A GFF3 field that describes the phase for CDS (coding sequence) annotations.

**score**

score: **undefined** | **number**

A GFF3 field that should describe the score of the annotation.

**seqid**

seqid: **undefined** | **string**

A GFF3 field: “The ID of the landmark used to establish the coordinate system for the current feature...”

**source**

source: **undefined** | **string**

A GFF3 field: “The source is a free text qualifier intended to describe the algorithm or operating procedure that generated this feature...”

**start**

start: **number**

The start coordinate of the Annotation.

**strand**

strand: **undefined** | Forward | Reverse | Unknown | Unoriented

A GFF3 field that describes the strand of the annotation.



**type**

```
type: undefined | string
```

A GFF3 field that is supposed to be “constrained to be either: (a) a term from the “lite” sequence ontology, SOFA; or (b) a SOFA accession number.” However, this is currently not enforced by SODA.

**3.2.23 GlyphConfig**

```
interface GlyphConfig<A extends Annotation, C extends Chart>
```

**Type parameters**

- A: Annotation
- C: Chart

**Properties****annotations**

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

**chart**

```
chart: C
```

The Chart object in which the glyphs will be rendered.

**fillColor**

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

**fillOpacity**

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

## height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

## initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

## selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

## strokeColor

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

## strokeDashArray

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

**strokeLineJoin**

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

**strokeOpacity**

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

**strokeWidth**

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

**target**

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

### x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

### y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

### zoomFn

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## 3.2.24 GlyphQueryConfig

```
interface GlyphQueryConfig
```

An interface that defines the parameters for a call to the queryGlyphMap() function.

### Properties

#### annotations

```
annotations: undefined | Annotation []
```

Constrain the query to these Annotations.

**chart**

```
chart: undefined | Chart <any>
```

Constrain the query to glyphs rendered in this Chart.

**selector**

```
selector: undefined | string
```

Constrain the query to glyphs with this selector.

**3.2.25 HighlightConfig**

```
interface HighlightConfig
```

This describes the parameters for a call to the Chart.highlight() function.

**Properties****color**

```
color: undefined | string
```

The color of the highlight. This defaults to black.

**end**

```
end: number
```

The end of the region to be highlighted in semantic coordinates.

**opacity**

```
opacity: undefined | number
```

The opacity of the highlight. This defaults to 0.1.

**selector**

```
selector: undefined | string
```

The selector that will be applied to the highlight object in the DOM. This will be auto generated if not supplied.

**start**

```
start: number
```

The start of the region to be highlighted in semantic coordinates.

### 3.2.26 HorizontalAxisConfig

```
interface HorizontalAxisConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the horizontalAxis rendering function.

**Type parameters**

- A: Annotation
- C: Chart

**Properties****annotations**

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

**axisType**

```
axisType: undefined | Bottom | Top
```

This determines whether the ticks and labels will be placed on the top or the bottom of the axis.

**chart**

```
chart: C
```

The Chart object in which the glyphs will be rendered.

### domain

```
domain: undefined | None | GlyphCallback <A, C, None>
```

This defines the domain of the D3 scale used to create the axis glyph.

### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

### fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

### fixed

```
fixed: undefined | boolean
```

If this is set to true, the axis glyph will not translate or scale during zoom events.

### height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

### initializeFn

```
initializeFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

**range**

range: **undefined** | None | GlyphCallback <A, C, None>

This defines the range of the D3 scale used to create the axis glyph.

**row**

row: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the row that the glyph is placed in.

**scaleToBinHeight**

scaleToBinHeight: **undefined** | **boolean**

If this is set to true, the axis glyph will be forced (by stretching) into the height of a row in the Chart.

**selector**

selector: **undefined** | **string**

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

**strokeColor**

strokeColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the color of the border around the glyph.

**strokeDashArray**

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>



**strokeDashOffset**

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

**strokeLineJoin**

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

**strokeOpacity**

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

**strokeWidth**

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

**target**

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### tickSizeOuter

```
tickSizeOuter: undefined | number | GlyphCallback <A, C, number>
```

This defines the tick property that will be passed to D3's `axis.tickSizeOuter` function. For more information, see [https://github.com/d3/d3-axis#axis\\_tickSizeOuter](https://github.com/d3/d3-axis#axis_tickSizeOuter)

### ticks

```
ticks: undefined | number | GlyphCallback <A, C, number>
```

This defines the tick property that will be passed to D3's `axis.ticks` function. For more information, see [https://github.com/d3/d3-axis#axis\\_ticks](https://github.com/d3/d3-axis#axis_ticks)

### width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

### x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

### y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

### zoomFn

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the `GlyphModifier` that will manage the glyphs created with this config. If provided, this callback function will override the `GlyphModifier`'s `zoom` method, which typically sets most of the positioning related properties from the `GlyphConfig`. Don't use this unless you know what you're doing.

### 3.2.27 HoverConfig

```
interface HoverConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the `hoverBehavior` function.

#### Type parameters

- A: Annotation
- C: Chart

#### Properties

##### annotations

```
annotations: A []
```

The Annotations to which the interaction is applied.

##### chart

```
chart: undefined | C
```

The Chart to which the interaction is applied.

##### mouseout

```
mouseout: InteractionCallback <A, C>
```

A callback function that will be responsible for executing the `mouseout` behavior. It receives a d3 selection of the glyph and the Annotation object it represents as arguments.

##### mouseover

```
mouseover: InteractionCallback <A, C>
```

A callback function that will be responsible for executing the `mouseover` behavior. It receives a d3 selection of the glyph and the Annotation object it represents as arguments.

### selector

```
selector: undefined | string
```

The selector of the glyphs to which the interaction is applied.

## 3.2.28 LineConfig

```
interface LineConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the line rendering function.

### Type parameters

- A: Annotation
- C: Chart

### Properties

#### annotations

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

#### chart

```
chart: C
```

The Chart object in which the glyphs will be rendered.

#### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

#### fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

## height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

## initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

## selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

## strokeColor

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

## strokeDashArray

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

### strokeWidth

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

### target

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

## width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

## x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

## x1

```
x1: undefined | number | GlyphCallback <A, C, number>
```

## x2

```
x2: undefined | number | GlyphCallback <A, C, number>
```

## y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

## y1

```
y1: undefined | number | GlyphCallback <A, C, number>
```

## y2

```
y2: undefined | number | GlyphCallback <A, C, number>
```

## zoomFn

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## 3.2.29 LinePlotConfig

```
interface LinePlotConfig<A extends PlotAnnotation, C extends Chart>
```

An interface that defines the parameters for a call to the linePlot rendering function.

### Type parameters

- A: PlotAnnotation
- C: Chart

## Properties

### annotations

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

### chart

```
chart: C
```

The Chart object in which the glyphs will be rendered.

### domain

```
domain: undefined | None | GlyphCallback <A, C, None>
```

This defines the domain of the plot.



**fillColor**

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

**fillOpacity**

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

**height**

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

**initializeFn**

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

**pathData**

```
pathData: undefined | string | GlyphCallback <A, C, string>
```

A callback that returns a string that defines the line's SVG path

**range**

```
range: undefined | None | GlyphCallback <A, C, None>
```

This defines the range of the plot.

**row**

row: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the row that the glyph is placed in.

**rowSpan**

rowSpan: **undefined** | **number**

The number of bins that the plot will span. This defaults to 1, which forces the plot to fit into one row. If an argument is supplied, it will cause the plot to grow downward. It will have no effect if a custom lineFunc is supplied.

**selector**

selector: **undefined** | **string**

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

**strokeColor**

strokeColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the color of the border around the glyph.

**strokeDashArray**

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

strokeDashOffset: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

strokeLineJoin: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

### strokeWidth

strokeWidth: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the width of the border around the glyph.

### target

target: **undefined** | Selection <any, any, any, any> | Viewport | Overflow | Defs

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

width: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel width of the glyph.

**x**

x: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel x coordinate of the glyph.

**y**

y: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel y coordinate of the glyph

**zoomFn**

zoomFn: **undefined** | (): **void**

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

### 3.2.30 MapVerticalLayout

**interface** MapVerticalLayout

An extension of VerticalLayout that additionally has a Map. This object is returned by SODA's default layout functions, and the Map is used to build the row() callback.

**Properties****row**

row: **GlyphCallback** <Annotation, Chart <any>, **number**>

This callback is used by default to place a glyph in a Chart's row.

**rowCount**

rowCount: **number**

This value is used by default to set the height of a Chart to display the required number of rows.

**rowMap**

```
rowMap: Map <string, number>
```

**3.2.31 PlotAnnotation**

```
interface PlotAnnotation
```

**Properties****end**

```
end: number
```

The end coordinate of the Annotation.

**id**

```
id: string
```

A unique identifier for the Annotation.

**start**

```
start: number
```

The start coordinate of the Annotation.

**values**

```
values: number []
```

**3.2.32 RadialChartConfig**

```
interface RadialChartConfig<P extends RenderParams>
```

A simple interface that defines the parameters that initialize a RadialChart

**Type parameters**

- P: RenderParams

## Properties

### axisType

axisType: **undefined** | Bottom | Top

This controls whether or not the Chart will render a horizontal axis.

### debugShading

debugShading: **undefined** | **boolean**

If this is set to true, the pad and viewport will be shaded so that they are visible in the browser.

### divHeight

divHeight: **undefined** | **string** | **number**

The height in pixels of the Chart's containing div.

### divMargin

divMargin: **undefined** | **string** | **number**

The CSS margin property for the Chart's div.

### divOutline

divOutline: **undefined** | **string**

The CSS outline property for the Chart's div.

### divOverflowX

divOverflowX: **undefined** | **string**

The CSS overflow-x setting of the Chart's containing div.

**divOverflowY**

divOverflowY: **undefined** | **string**

The CSS overflow-y setting of the Chart's containing div.

**divWidth**

divWidth: **undefined** | **string** | **number**

The width in pixels of the Chart's containing div.

**domainConstraint**

domainConstraint: **undefined** | (chart: **Chart** <P>): **None**

This constrains the Chart's domain, which in turn constrains both zoom level and panning. The parameter is a callback function that is evaluated after each zoom event to produce an interval that constrains the domain.

**draw**

draw: **undefined** | (params: **P**): **void**

The rendering callback that should be responsible for drawing glyphs with the rendering API.

**id**

id: **undefined** | **string**

A unique identifier for the Chart. This will be generated automatically if one isn't provided.

**leftPadSize**

leftPadSize: **undefined** | **number**

The number of pixels of padding on the left side of the Chart.

**lowerPadSize**

lowerPadSize: **undefined** | **number**

The number of pixels of padding on the bottom of the Chart.

**padSize**

padSize: **undefined** | **number**

The number of pixels of padding around each edge of the Chart.

**postRender**

postRender: **undefined** | (params: **P**): **void**

The callback function that the Chart executes after render() is called.

**postResize**

postResize: **undefined** | (): **void**

The callback function that the Chart executes after resize() is called.

**postZoom**

postZoom: **undefined** | (): **void**

The callback function that the Chart executes after zoom() is called.

**resizable**

resizable: **undefined** | **boolean**

This controls whether or not the Chart will automatically resize itself as it's container changes size. This will cause the Chart to ignore explicit height/width arguments in the config.



### rightPadSize

```
rightPadSize: undefined | number
```

The number of pixels of padding on the right side of the Chart.

### rowColors

```
rowColors: undefined | string []
```

A list of colors that will color the Chart's rows in a repeating pattern.

### rowCount

```
rowCount: undefined | number
```

The number of rows that will be rendered.

### rowHeight

```
rowHeight: undefined | number
```

The height in pixels of a horizontal row in the Chart. This defaults to a value of 10.

### rowOpacity

```
rowOpacity: undefined | number
```

The opacity of the colored row stripes.

### selector

```
selector: string
```

A string that can be used to uniquely select the target DOM container.

### trackHeight

```
trackHeight: undefined | number
```

The “height” of the radial track on which annotations will be rendered. Conceptually, this is equal to the difference of the radii of two concentric circles that define an annulus.

### **updateDimensions**

updateDimensions: **undefined** | (params: **P**): **void**

The rendering callback function that should be responsible for updating the Chart's DOM element dimensions.

### **updateDomain**

updateDomain: **undefined** | (params: **P**): **void**

The rendering callback function that should be responsible for updating the domain of the Chart.xScale property.

### **updateLayout**

updateLayout: **undefined** | (params: **P**): **void**

The rendering callback function that should be responsible for updating the Chart.layout property.

### **updateRowCount**

updateRowCount: **undefined** | (params: **P**): **void**

The rendering callback function that should be responsible for updating the Chart.rowCount property.

### **upperPadSize**

upperPadSize: **undefined** | **number**

The number of pixels of padding on the top of the Chart.

### **zoomConstraint**

zoomConstraint: **undefined** | **None**

A Chart's contents are scaled by a scaling factor  $k$ . If a zoomConstraint of the form  $[\text{min\_}k, \text{max\_}k]$  is provided, the scaling factor will be constrained to that interval. This will not constrain panning.

## zoomable

zoomable: **undefined** | **boolean**

This controls whether or not the Chart will be configured to allow zooming and panning.

### 3.2.33 RectangleConfig

**interface** RectangleConfig<A **extends** Annotation, C **extends** Chart>

An interface that defines the parameters for a call to the rectangle rendering function.

#### Type parameters

- A: Annotation
- C: Chart

#### Properties

##### annotations

annotations: **A** []

A list of Annotation objects that will be used to render the glyphs.

##### chart

chart: **C**

The Chart object in which the glyphs will be rendered.

##### fillColor

fillColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the fill color of the glyph.

##### fillOpacity

fillOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the fill opacity of the glyph.

## height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

## initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

## selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

## strokeColor

```
strokeColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the color of the border around the glyph.

## strokeDashArray

```
strokeDashArray: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

```
strokeDashOffset: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

```
strokeLineCap: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

**strokeLineJoin**

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

**strokeOpacity**

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

**strokeWidth**

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

**target**

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### width

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

### x

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

### y

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

### zoomFn

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## 3.2.34 RenderParams

```
interface RenderParams
```

This defines the parameters for a call to a Chart's rendering method.

### Properties

#### annotations

```
annotations: undefined | Annotation []
```

The Annotation objects to be rendered.

**end**

end: **undefined** | **number**

The end coordinate of the region that will be rendered.

**rowCount**

rowCount: **undefined** | **number**

The number of rows that will be rendered.

**start**

start: **undefined** | **number**

The start coordinate of the region that will be rendered.

### 3.2.35 SequenceAnnotation

**interface** SequenceAnnotation

**Properties****end**

end: **number**

The end coordinate of the Annotation.

**id**

id: **string**

A unique identifier for the Annotation.

**sequence**

sequence: **string**

**start**

start: **number**

The start coordinate of the Annotation.

### 3.2.36 SequenceConfig

**interface** SequenceConfig<S **extends** SequenceAnnotation, C **extends** Chart>

An interface that defines the parameters for a call to the sequence rendering function.

**Type parameters**

- S: SequenceAnnotation
- C: Chart

**Properties****annotations**

annotations: **S** []

A list of Annotation objects that will be used to render the glyphs.

**chart**

chart: **C**

The Chart object in which the glyphs will be rendered.

**fillColor**

fillColor: **undefined** | **string** | GlyphCallback <S, C, **string**>

A callback to define the fill color of the glyph.

**fillOpacity**

fillOpacity: **undefined** | **number** | GlyphCallback <S, C, **number**>

A callback to define the fill opacity of the glyph.



## height

```
height: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the pixel height of the glyph.

## initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

## row

```
row: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the row that the glyph is placed in.

## selector

```
selector: undefined | string
```

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

## strokeColor

```
strokeColor: undefined | string | GlyphCallback <S, C, string>
```

A callback to define the color of the border around the glyph.

## strokeDashArray

```
strokeDashArray: undefined | string | GlyphCallback <S, C, string>
```

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

```
strokeDashOffset: undefined | string | GlyphCallback <S, C, string>
```

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

```
strokeLineCap: undefined | string | GlyphCallback <S, C, string>
```

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

```
strokeLineJoin: undefined | string | GlyphCallback <S, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

```
strokeOpacity: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the opacity of the border around the glyph.

### strokeWidth

```
strokeWidth: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the width of the border around the glyph.

### target

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

**width**

```
width: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the pixel width of the glyph.

**x**

```
x: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the pixel x coordinate of the glyph.

**y**

```
y: undefined | number | GlyphCallback <S, C, number>
```

A callback to define the pixel y coordinate of the glyph

**zoomFn**

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

**3.2.37 SimpleTextConfig**

```
interface SimpleTextConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the text rendering function.

**Type parameters**

- A: Annotation
- C: Chart

**Properties****alignmentBaseline**

```
alignmentBaseline: undefined | string | GlyphCallback <A, C, string>
```

How the text glyph is aligned with it's parent. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/alignment-baseline>

## annotations

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

## chart

```
chart: C
```

The Chart object in which the glyphs will be rendered.

## fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

## fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

## fontFamily

```
fontFamily: undefined | string | GlyphCallback <A, C, string>
```

The font family that will be used. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-family>

## fontSize

```
fontSize: undefined | number | GlyphCallback <A, C, number>
```

The font size of the text.

## fontStyle

```
fontStyle: undefined | string | GlyphCallback <A, C, string>
```

The font style: normal, italic, or oblique. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-style>

### fontWeight

fontWeight: **undefined** | **string** | GlyphCallback <A, C, **string**>

The weight of the font: normal, bold, bolder, lighter. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-weight>

### height

height: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the pixel height of the glyph.

### initializeFn

initializeFn: **undefined** | (): **void**

### row

row: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the row that the glyph is placed in.

### selector

selector: **undefined** | **string**

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

### strokeColor

strokeColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the color of the border around the glyph.

**strokeDashArray**

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

**strokeDashOffset**

strokeDashOffset: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

**strokeLineCap**

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

**strokeLineJoin**

strokeLineJoin: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

**strokeOpacity**

strokeOpacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the opacity of the border around the glyph.

**strokeWidth**

strokeWidth: **undefined** | **number** | GlyphCallback <A, C, **number**>

A callback to define the width of the border around the glyph.

## target

target: <b>undefined</b>   Selection <any, any, any, any>   Viewport   Overflow   Defs
--

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

## text

text: <b>GlyphProperty</b> <A, C, <b>string</b> >
---

The text to display in the glyph.

## textAnchor

textAnchor: <b>undefined</b>   <b>string</b>   GlyphCallback <A, C, <b>string</b> >
---

Where the text is aligned to: start, middle, or end. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/text-anchor>

## width

width: <b>undefined</b>   <b>number</b>   GlyphCallback <A, C, <b>number</b> >
--

A callback to define the pixel width of the glyph.

## x

x: <b>undefined</b>   <b>number</b>   GlyphCallback <A, C, <b>number</b> >
--

A callback to define the pixel x coordinate of the glyph.

## y

y: <b>undefined</b>   <b>number</b>   GlyphCallback <A, C, <b>number</b> >
--

A callback to define the pixel y coordinate of the glyph

## zoomFn

```
zoomFn: undefined | (): void
```

### 3.2.38 SliceConfig

```
interface SliceConfig<A extends Annotation>
```

#### Type parameters

- A: Annotation

#### Properties

##### annotations

```
annotations: A []
```

##### end

```
end: number
```

##### start

```
start: number
```

### 3.2.39 TooltipConfig

```
interface TooltipConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the tooltip function.

#### Type parameters

- A: Annotation
- C: Chart



## Properties

### annotations

```
annotations: A []
```

The Annotations to which the interaction is applied.

### backgroundColor

```
backgroundColor: undefined | string | GlyphCallback <A, C, string>
```

This defines the background color of the tooltip.

### borderRadius

```
borderRadius: undefined | number | GlyphCallback <A, C, number>
```

This defines the border radius of the tooltip.

### chart

```
chart: undefined | C
```

The Chart to which the interaction is applied.

### fontFamily

```
fontFamily: undefined | string | GlyphCallback <A, C, string>
```

The font family that will be used. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-family>

### fontSize

```
fontSize: undefined | number | GlyphCallback <A, C, number>
```

The font size of the text.

**fontStyle**

fontStyle: **undefined** | **string** | GlyphCallback <A, C, **string**>

The font style: normal, italic, or oblique. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-style>

**fontWeight**

fontWeight: **undefined** | **string** | GlyphCallback <A, C, **string**>

The weight of the font: normal, bold, bolder, lighter. See: <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/font-weight>

**opacity**

opacity: **undefined** | **number** | GlyphCallback <A, C, **number**>

This defines the opacity of the tooltip.

**padding**

padding: **undefined** | **number** | GlyphCallback <A, C, **number**>

This defines the CSS padding of the tooltip.

**selector**

selector: **undefined** | **string**

The selector of the glyphs to which the interaction is applied.

**text**

text: **GlyphProperty** <A, C, **string**>

This defines the text for the tooltip.

**textColor**

```
textColor: undefined | string | GlyphCallback <A, C, string>
```

This defines the tooltip text color.

**3.2.40 VerticalAxisConfig**

```
interface VerticalAxisConfig<A extends Annotation, C extends Chart>
```

An interface that defines the parameters for a call to the verticalAxis rendering function.

**Type parameters**

- A: Annotation
- C: Chart

**Properties****annotations**

```
annotations: A []
```

A list of Annotation objects that will be used to render the glyphs.

**axisType**

```
axisType: undefined | Left | Right
```

This determines whether the ticks and labels will be placed on the left or the right of the axis.

**chart**

```
chart: C
```

The Chart object in which the glyphs will be rendered.

**domain**

```
domain: undefined | None | GlyphCallback <A, C, None>
```

This defines the domain of the axis.

### fillColor

```
fillColor: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the fill color of the glyph.

### fillOpacity

```
fillOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the fill opacity of the glyph.

### height

```
height: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel height of the glyph.

### initializeFn

```
initializeFn: undefined | () : void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's initialization method, which typically sets most of the style related properties from the GlyphConfig. Don't use this unless you know what you're doing.

### range

```
range: undefined | None | GlyphCallback <A, C, None>
```

This defines the range of the axis.

### row

```
row: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the row that the glyph is placed in.

### rowSpan

rowSpan: **undefined** | **number**

The number of bins that the axis will span. This defaults to 1, which forces the axis to fit into one row. If an argument is supplied, it will cause the axis to grow downward. It will have no effect if a custom domain function is supplied.

### selector

selector: **undefined** | **string**

The string that will be used to uniquely identify the call to the glyph rendering function. In the DOM, the glyphs' elements will have this assigned as an ID. If the same selector is supplied to two distinct calls to the same glyph function, the rendering results of the first call will be cleared and replaced with the results of the second.

### strokeColor

strokeColor: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the color of the border around the glyph.

### strokeDashArray

strokeDashArray: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke dash array of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dasharray>

### strokeDashOffset

strokeDashOffset: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the offset for the stroke dash array (if supplied) of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-dashoffset>

### strokeLineCap

strokeLineCap: **undefined** | **string** | GlyphCallback <A, C, **string**>

A callback to define the stroke linecap of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linecap>

### strokeLineJoin

```
strokeLineJoin: undefined | string | GlyphCallback <A, C, string>
```

A callback to define the offset for the stroke linejoin of the glyph. See <https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin>

### strokeOpacity

```
strokeOpacity: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the opacity of the border around the glyph.

### strokeWidth

```
strokeWidth: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the width of the border around the glyph.

### target

```
target: undefined | Selection <any, any, any, any> | Viewport | Overflow | Defs
```

This determines the parent DOM element in which the glyphs will be rendered. When supplying a BindTarget, the rendering function will find the appropriate parent in the supplied Chart. When supplying a D3 selection, the rendering function will explicitly use the selected element.

### tickSizeOuter

```
tickSizeOuter: undefined | number | GlyphCallback <A, C, number>
```

This defines the tick property that will be passed to D3's axis.tickSizeOuter function. For more information, see [https://github.com/d3/d3-axis#axis\\_tickSizeOuter](https://github.com/d3/d3-axis#axis_tickSizeOuter)

### ticks

```
ticks: undefined | number | GlyphCallback <A, C, number>
```

This defines the tick property that will be passed to D3's axis.ticks function. For more information, see [https://github.com/d3/d3-axis#axis\\_ticks](https://github.com/d3/d3-axis#axis_ticks)

**width**

```
width: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel width of the glyph.

**x**

```
x: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel x coordinate of the glyph.

**y**

```
y: undefined | number | GlyphCallback <A, C, number>
```

A callback to define the pixel y coordinate of the glyph

**zoomFn**

```
zoomFn: undefined | (): void
```

A callback function that will be passed to the GlyphModifier that will manage the glyphs created with this config. If provided, this callback function will override the GlyphModifier's zoom method, which typically sets most of the positioning related properties from the GlyphConfig. Don't use this unless you know what you're doing.

**3.2.41 VerticalLayout**

```
interface VerticalLayout
```

An interface that defines the object that Charts use to store the vertical layout of glyphs.

**Properties****row**

```
row: GlyphCallback <Annotation, Chart <any>, number>
```

This callback is used by default to place a glyph in a Chart's row.

## rowCount

`rowCount: number`

This value is used by default to set the height of a Chart to display the required number of rows.

## 3.3 Functions

### 3.3.1 aggregateIntransitive

`function aggregateIntransitive<A extends Annotation>(config: AggregationConfig <A>): None`

A utility function that aggregates Annotation objects into Annotation groups based off of the supplied criterion. This function assumes that your aggregation criterion is not transitive, i.e. if criterion(a, b) and criterion(b, c) evaluate to true, then criterion(a, c) doesn't necessarily evaluate to true.

#### Type parameters

- A: Annotation

#### Parameters

- config: AggregationConfig <A>

**Returns:** AnnotationGroup <A> []

### 3.3.2 aggregateTransitive

`function aggregateTransitive<A extends Annotation>(config: AggregationConfig <A>): None`

A utility function that aggregates Annotation objects into Annotation groups based off of the supplied criterion. This function assumes that your aggregation criterion is transitive, i.e. if criterion(a, b) and criterion(b, c) evaluate to true, then criterion(a, c) must evaluate to true.

#### Type parameters

- A: Annotation

#### Parameters

- config: AggregationConfig <A>

**Returns:** AnnotationGroup <A> []

### 3.3.3 arc

`function arc<A extends Annotation, C extends Chart>(config: ArcConfig <A, C>): d3.  
↪ Selection`

This renders a list of Annotation objects as arcs in a Chart.

#### Type parameters

- A: Annotation



- C: Chart

#### Parameters

- config: ArcConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.4 area

```
function area<A extends PlotAnnotation, C extends Chart>(config: AreaConfig <A, C>): d3.
↳ Selection
```

This renders PlotAnnotations as area glyphs in a Chart.

#### Type parameters

- A: PlotAnnotation
- C: Chart

#### Parameters

- config: AreaConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.5 augment

```
function augment<T extends generic>(config: AugmentConfig <T>): None
```

This takes a list of any object T, and a set of callback functions that describe how to give it the id, start, and end properties that satisfy the Annotation interface. The idea here is to allow you to get valid Annotation objects without having to write a class. Each property function is wrapped in an AugmentParam object has one other boolean property called “virtual.” If virtual is set to true, the callback function will be applied as a getter for its corresponding property. If virtual is false or undefined, the callback function will be evaluated while augment() is running and the value will be applied as a real property on the object. Finally, the augment function checks to make sure that each Annotation property on each object has the correct type, throwing an exception if there are any incorrect types. This check can be skipped by setting skipValidate to true, probably improving performance measurably. You’ll want to be careful if you decide to skip the validation, and if you’re really worried about performance you’ll probably want to avoid using this function altogether and write a proper class.

#### Type parameters

- T: generic

#### Parameters

- config: AugmentConfig <T>

**Returns:** T & Annotation []

### 3.3.6 axisRadialInner

```
function axisRadialInner(angleScale: d3.ScaleLinear <number, number>, radius: number):   
↳ None
```

#### Parameters

- angleScale: d3.ScaleLinear <number, number>
- radius: number

**Returns:** (selection: d3.Selection <any, any, any, any>): void

### 3.3.7 axisRadialOuter

```
function axisRadialOuter(angleScale: d3.ScaleLinear <number, number>, radius: number):   
↳ None
```

#### Parameters

- angleScale: d3.ScaleLinear <number, number>
- radius: number

**Returns:** (selection: d3.Selection <any, any, any, any>): void

### 3.3.8 barPlot

```
function barPlot<A extends PlotAnnotation, C extends Chart>(config: BarPlotConfig <A, C>  
↳): d3.Selection
```

This renders PlotAnnotations as bar plots in a Chart.

#### Type parameters

- A: PlotAnnotation
- C: Chart

#### Parameters

- config: BarPlotConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.9 chevronLine

```
function chevronLine<A extends Annotation, C extends Chart>(config: ChevronLineConfig <A,  
↳ C>): d3.Selection
```

This renders Annotations as lines with chevron arrows in a Chart.

#### Type parameters

- A: Annotation
- C: Chart

**Parameters**

- config: ChevronLineConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

**3.3.10 chevronRectangle**

```
function chevronRectangle<A extends Annotation, C extends Chart>(config: ChevronRectangleConfig <A, C>): d3.Selection
```

This renders Annotations as rectangles with chevron arrows in a Chart.

**Type parameters**

- A: Annotation
- C: Chart

**Parameters**

- config: ChevronRectangleConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

**3.3.11 clickBehavior**

```
function clickBehavior<A extends Annotation, C extends Chart>(config: ClickConfig <A, C>): void
```

This applies click interactions to a list of Annotations.

**Type parameters**

- A: Annotation
- C: Chart

**Parameters**

- config: ClickConfig <A, C>

**Returns:** void

**3.3.12 dynamicText**

```
function dynamicText<A extends Annotation, C extends Chart>(config: DynamicTextConfig <A, C>): d3.Selection
```

This renders a list of Annotation objects as text in a Chart.

**Type parameters**

- A: Annotation
- C: Chart

**Parameters**

- config: DynamicTextConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.13 exportPng

```
function exportPng<C extends Chart>(config: ExportConfig <C>): void
```

Save the current view in a chart as a PNG image.

**Type parameters**

- C: Chart

**Parameters**

- config: ExportConfig <C>

**Returns:** void

### 3.3.14 generateAnnotations

```
function generateAnnotations(conf: AnnotationGenerationConfig): None
```

A utility function to generate some uniformly distributed Annotation objects. This is intended for testing/prototyping/playing around.

**Parameters**

- conf: AnnotationGenerationConfig

**Returns:** Annotation []

### 3.3.15 generateId

```
function generateId(prefix: string): string
```

Get an auto-generated string identifier of the form “<prefix>-<count>,” where prefix defaults to “soda-id” and count is incremented for every call to this function. A unique count is maintained for each prefix.

**Parameters**

- prefix: string

**Returns:** string

### 3.3.16 generatePlotAnnotations

```
function generatePlotAnnotations(conf: AnnotationGenerationConfig): None
```

A utility function to generate some PlotAnnotation objects. This is intended for testing/prototyping/playing around.

**Parameters**

- conf: AnnotationGenerationConfig

**Returns:** PlotAnnotation []

### 3.3.17 generateSequenceAnnotations

```
function generateSequenceAnnotations(conf: AnnotationGenerationConfig): None
```

A utility function to generate some SequenceAnnotation objects. This is intended for testing/prototyping/playing around.

#### Parameters

- conf: AnnotationGenerationConfig

**Returns:** SequenceAnnotation []

### 3.3.18 getAlignmentAnnotations

```
function getAlignmentAnnotations(config: AlignmentConfig): None
```

This returns a set of SequenceAnnotations defined such that the provided query sequence can be rendered in a Chart as if it were aligned to a chromosome. The matches, substitutions, gaps, and insertions are returned as separate objects. The idea here is that they can be rendered individually with different style parameters.

#### Parameters

- config: AlignmentConfig

**Returns:** None

### 3.3.19 getAllAnnotationIds

```
function getAllAnnotationIds(): None
```

This returns a list of all of the Annotation IDs that have been used to render glyphs.

**Returns:** string []

### 3.3.20 getAnnotationById

```
function getAnnotationById(id: string): Annotation
```

This function produces a reference to Annotation object that is mapped with the provided string id. It will throw an exception if the id is not in the internal map.

#### Parameters

- id: string

**Returns:** Annotation

### 3.3.21 getAxis

```
function getAxis(scale: d3.ScaleLinear <number, number>, axisType: AxisType): d3.Axis
```

A utility function that returns the results of the various d3 axis functions.

#### Parameters

- scale: d3.ScaleLinear <number, number>
- axisType: AxisType

**Returns:** d3.Axis <number | None>

### 3.3.22 greedyGraphLayout

```
function greedyGraphLayout<A extends Annotation>(ann: A [], tolerance: number, ↵  
↵vertSortFunction: (verts: string [], graph: AnnotationGraph <A>): void): ↵  
↵MapVerticalLayout
```

This function takes a list of Annotation objects and uses a deterministic greedy graph coloring algorithm to assign each of them a y coordinate in terms of horizontal bins that will prevent any horizontal overlap when they are rendered in a Chart.

#### Type parameters

- A: Annotation

#### Parameters

- ann: A []
- tolerance: number
- vertSortFunction: (verts: string [], graph: AnnotationGraph <A>): void

**Returns:** MapVerticalLayout

### 3.3.23 heatmap

```
function heatmap<A extends PlotAnnotation, C extends Chart>(config: HeatmapConfig <A, C>  
↵): d3.Selection
```

This renders PlotAnnotations as heatmaps in a Chart.

#### Type parameters

- A: PlotAnnotation
- C: Chart

#### Parameters

- config: HeatmapConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.24 heuristicGraphLayout

```
function heuristicGraphLayout(ann: Annotation [], nIters: number, tolerance: number): ↪ MapVerticalLayout
```

This function takes a list of Annotation objects and uses a non-deterministic greedy graph coloring heuristic to assign each of them a y coordinate in terms of horizontal bins that will prevent any horizontal overlap when they are rendered in a Chart.

#### Parameters

- ann: Annotation []
- nIters: number
- tolerance: number

**Returns:** MapVerticalLayout

### 3.3.25 horizontalAxis

```
function horizontalAxis<A extends Annotation, C extends Chart>(config: ↪ HorizontalAxisConfig <A, C>): d3.Selection
```

This renders Annotations as horizontal axes in a Chart.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- config: HorizontalAxisConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.26 hoverBehavior

```
function hoverBehavior<A extends Annotation, C extends Chart>(config: HoverConfig <A, C> ↪): void
```

This applies hover interactions to a list of Annotations.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- config: HoverConfig <A, C>

**Returns:** void

### 3.3.27 intervalGraphLayout

```
function intervalGraphLayout(ann: Annotation [], tolerance: number): MapVerticalLayout
```

This function takes a list of Annotation objects and uses a greedy interval scheduling algorithm to assign each of them a y coordinate in terms of horizontal bins that will prevent any horizontal overlap when they are rendered in a Chart.

#### Parameters

- ann: Annotation []
- tolerance: number

**Returns:** MapVerticalLayout

### 3.3.28 line

```
function line<A extends Annotation, C extends Chart>(config: LineConfig <A, C>): d3.  
↳Selection
```

This renders a list of Annotation objects as lines in a Chart.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- config: LineConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.29 linePlot

```
function linePlot<A extends PlotAnnotation, C extends Chart>(config: LinePlotConfig <A, C>): d3.Selection
```

This renders PlotAnnotations as line plots in a Chart.

#### Type parameters

- A: PlotAnnotation
- C: Chart

#### Parameters

- config: LinePlotConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>



### 3.3.30 parseBedRecords

```
function parseBedRecords(records: string | string []): None
```

A utility function to parse a general BED record. There are no guarantees about which fields end up being present in the resulting BED objects.

#### Parameters

- records: string | string []

**Returns:** BedAnnotation []

### 3.3.31 parseGff3Records

```
function parseGff3Records(records: string | string []): None
```

A utility function to parse a GFF3 records. This function accepts either a string of newline delimited GFF3 records, or an array of individual record strings.

#### Parameters

- records: string | string []

**Returns:** Gff3Annotation []

### 3.3.32 parseOrientation

```
function parseOrientation(str: string): Orientation
```

A utility function to parse an Orientation enum from a string. For now, this is pretty basic and far from robust.

#### Parameters

- str: string

**Returns:** Orientation

### 3.3.33 queryGlyphMap

```
function queryGlyphMap(config: GlyphQueryConfig): None
```

This function returns GlyphMappings. If all three parameters (id, selector, chart) are supplied in the config, the function will return a single D3 selection. Otherwise, the function will return a list of D3 selections.

#### Parameters

- config: GlyphQueryConfig

**Returns:** d3.Selection [] | undefined

### 3.3.34 radialRectangle

```
function radialRectangle<A extends Annotation, C extends RadialChart>(config: ↵  
↵RectangleConfig <A, C>): d3.Selection
```

This renders a list of Annotation objects as rectangles in a RadialChart.

#### Type parameters

- A: Annotation
- C: RadialChart

#### Parameters

- config: RectangleConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.35 rectangle

```
function rectangle<A extends Annotation, C extends Chart>(config: RectangleConfig <A, C>  
↵): d3.Selection
```

This renders a list of Annotation objects as rectangles in a Chart.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- config: RectangleConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.36 removeGlyphsByQuery

```
function removeGlyphsByQuery(config: GlyphQueryConfig): void
```

#### Parameters

- config: GlyphQueryConfig

**Returns:** void

### 3.3.37 resolveValue

```
function resolveValue<A extends Annotation, C extends Chart, V extends generic>  
↵(property: GlyphProperty <A, C, V>, d: AnnotationDatum <A, C>): V
```

A utility function that resolves the value from a GlyphProperty. If the property is a callback function, it will be called to retrieve the value. Otherwise, it will just return the value.

#### Type parameters

- A: Annotation
- C: Chart
- V: generic

**Parameters**

- property: GlyphProperty <A, C, V>
- d: AnnotationDatum <A, C>

**Returns:** V

### 3.3.38 sequence

```
function sequence<S extends SequenceAnnotation, C extends Chart>(config: SequenceConfig
↳<S, C>): d3.Selection
```

This renders a list of SequenceAnnotation objects as sequence glyphs in a Chart.

**Type parameters**

- S: SequenceAnnotation
- C: Chart

**Parameters**

- config: SequenceConfig <S, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.39 setKeySeparator

```
function setKeySeparator(separator: string): void
```

Set the separator that SODA uses to build map keys. The keys are of the form: <annotation ID><separator><glyph selector><separator><chart ID>.

**Parameters**

- separator: string

**Returns:** void

### 3.3.40 simpleText

```
function simpleText<A extends Annotation, C extends Chart>(config: SimpleTextConfig <A, C>): d3.Selection
```

This renders a list of Annotation objects as text in a Chart.

**Type parameters**

- A: Annotation
- C: Chart

**Parameters**

- config: SimpleTextConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

### 3.3.41 slicePlotAnnotations

```
function slicePlotAnnotations(config: SliceConfig <PlotAnnotation>): None
```

#### Parameters

- config: SliceConfig <PlotAnnotation>

**Returns:** AnnotationGroup | undefined

### 3.3.42 sliceSequenceAnnotations

```
function sliceSequenceAnnotations(config: SliceConfig <SequenceAnnotation>): None
```

#### Parameters

- config: SliceConfig <SequenceAnnotation>

**Returns:** AnnotationGroup | undefined

### 3.3.43 tooltip

```
function tooltip<A extends Annotation, C extends Chart>(config: TooltipConfig <A, C>):  
↳ void
```

This applies tooltip interactions to a list of Annotations.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- config: TooltipConfig <A, C>

**Returns:** void

### 3.3.44 unmapAnnotationById

```
function unmapAnnotationById(id: string): void
```

#### Parameters

- id: string

**Returns:** void

### 3.3.45 verticalAxis

```
function verticalAxis<A extends Annotation, C extends Chart>(config: VerticalAxisConfig
  ↳<A, C>): d3.Selection
```

This renders Annotations as vertical axes in a chart. This is intended to be used in conjunction with one of the plotting glyph modules.

#### Type parameters

- A: Annotation
- C: Chart

#### Parameters

- config: VerticalAxisConfig <A, C>

**Returns:** d3.Selection <SVGGElement, string, any, any>

## 3.4 Enumerations

### 3.4.1 AxisType

```
enum AxisType
```

A simple enum to serve as an argument for selecting which D3 Axis function to call.

#### Members

##### Bottom

```
Bottom: = "bottom"
```

##### Left

```
Left: = "left"
```

##### Right

```
Right: = "right"
```

## Top

```
Top: = "top"
```

### 3.4.2 BindTarget

```
enum BindTarget
```

An enumeration of the targets in a Chart that an Annotation can be bound to.

#### Members

##### Defs

```
Defs: = "defs"
```

The defs section, where things like patterns are supposed to go.

##### Overflow

```
Overflow: = "overflow"
```

The secondary viewport of a Chart in which a glyph is allowed to render outside the explicit bounds.

##### Viewport

```
Viewport: = "viewport"
```

The default viewport of a Chart.

### 3.4.3 FillDirection

```
enum FillDirection
```

A simple enum to define the direction that an area glyph fills in.

#### Members

##### Down

```
Down: = "down"
```

## Up

```
Up: = "up"
```

### 3.4.4 GenerationPattern

```
enum GenerationPattern
```

#### Members

##### Random

```
Random: = "random"
```

##### Sequential

```
Sequential: = "sequential"
```

### 3.4.5 Orientation

```
enum Orientation
```

A simple enum to define strand orientation.

#### Members

##### Forward

```
Forward: = "+"
```

Represents the forward strand.

##### Reverse

```
Reverse: = "-"
```

Represents the reverse strand.

### Unknown

`Unknown: = "?"`

Represents an unknown strand where strand information would be relevant (if it were known).

### Unoriented

`Unoriented: = "."`

Represents no strand.

This is the documentation for SODA, a TypeScript/Javascript library for creating genomic annotation visualizations.