D3 Cheatsheet
by Blair Labatt III, page 1 of 2

D3 is rigorously declarative, but not purely functional. Most of the work is done in stateful "function objects", of which ∃ 4 main types:

- Factory$_F$: implement prescribed APIs
- Generator$_G$: Generate concrete visualization code (SVG, Canvas) from passed data
- Layout$_L$: Transform passed dataset to include additional visual layout information
- Component$_C$: Manipulate the DOM

Subscripts are used herein to categorize D3 functions according to the above taxonomy.

1 Data Joining & Selections

Selecting [d3]

Create a selection with one of the following top-level calls, generally passing a W3C selector string:

selection	select	selectAll
selector	selectorAll	matcher
window	style	

Create derivative selections (subsets, unions, nestings) by invoking one of the following methods of an already-existing selection:

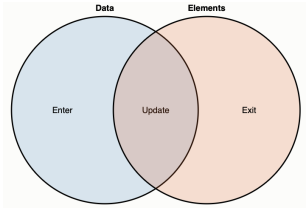
select selectAll filter merge

Modifying [<selection>]

Change properties of DOM elements in bulk:

attr	classed	property
text	html	append
insert	remove	clone
sort	order	raise
lower		

Joining [<selection>]



The "General Update Pattern" involves a "data join", followed by references to the resulting subset of elements & data, and looks something like this:

```
svg.selectAll("circle")
   .data(data)
   .enter().append("circle")
      .attr("cx", function(d) { return d.x; })
      .attr("cy", function(d) { return d.y; })
      .attr("r", 2.5);
```

data join enter exit datum

Events [d3]

Add/remove an event-handler with `<selection>.on()`*, or immediately dispatch an event with* `<selection>.dispatch()`*. The following top-level functions return information about an active event:*

event	customEvent	mouse
touch	touches	clientPoint

Control [<selection>]

The following yield information about selections, except for `each` *and* `call`*, which afford arbitrary code execution for selections while maintaining the ability to chain subsequent methods.*

each	call	empty
nodes	node	size

Local Variables [<selection>]

Afford the storage/retrieval of state that is independent of a selection's data.

set get remove toString

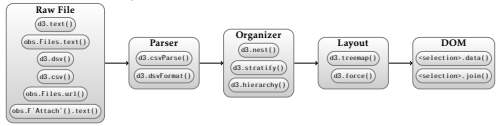
Namespaces [d3]

For use in construction XML namespaces.

namespace namespaces

2 Munging & Formatting

Data importation process can be complex. Here is a somewhat general pattern:



Delimiter-Separated [<dsv>]

[De-]Serialize raw files. (Can alternatively use concomitant command-line tool.)

parse	parseRows	format
formatBody	formatRows	formatRow
formatValue	autoType	

```
d3.dsvFormat("|").parse("foo|bar\n1|2");
```

For CSV, TSV, ∃ top-level shorthands for the above:

```
d3.csvParse("foo,bar\n1,2");
d3.tsvParse("foo\tbar\n1\t2");
d3.csvFormat([{foo: "1", bar: "2"}]);
d3.tsvFormat([{foo: "1", bar: "2"}]);
```

Arrays [d3]

Extending the already-extensive set of native javascript array functions, these top-level methods take an array and yield transformations or statistical meta-information.

min	max	extent
sum	mean	median
variance	deviation	cumsum
bisect	bisectLeft	fsum
bisector	quickselect	scan
quantile	ascending	bisectRight
cross	merge	descending
permute	shuffle	pairs
tickIncrement	tickStep	ticks
transpose	least	range
least index[es]	group	groups
	rollup[s]	zip

Can also sub-divide array data into bins using `d3.histogram()` *and its methods.*

Hierarchies [<node>]

Format data into nested, tree-like structure. Create with top-level call `d3.hierarchy()`*, which returns parent node.*

ancestors	descendents	leaves
path	links	sum
count	sort	each
eachAfter	eachBefore	copy

`d3.stratify` *transforms data from "link format" into a proper* `hierarchy`*.*

Number Formats

Pass a `specifier` *string to a formatter, eg:*

d3.format(".0%")(0.123);	% 12%
d3.format("($.2f")(-3.5);	% (£3.50)
d3.format(".2s")(42e6);	% 42M
d3.format("#x")(48879);	% 0xbeef
d3.format(",.2r")(4223);	% 4,200

...where `specifier` *takes the form:*

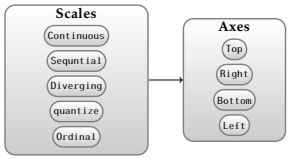
```
[[fill]align][sign][symbol][0]
[width][,][.precision][~][type]
```

Serialize specifiers with `d3.formatSpecifier`*. Create formatter for non-default "locale" with* `d3.formatLocale`*. Add SI Unit prefixes with* `<locale>.formatPrefix`*:*

```
d3.formatPrefix(",.0", 1e-6)(.00042);  % 420µ
```

Time Formatting & Calc [d3.time-]

3 Scales & Axes



See also here for a lovely visual explanation.

Continuous [<scale>]

Sub-types: power^p, log^l, time^t:

invert	interpolate	copy
domain	unknown	exponent^p
range	ticks	base^l
rangeRound	tickFormat	
clamp	nice	

Sequential [d3.scaleSquential-]

Log	Sqrt	Symlog	Quantile
Pow			

Diverging [d3.scaleDiverging]

Log Pow Sqrt Symlog

Quantize [<scale>]

Sub-types: quantile^l, quantize^z, threshold^t:

invertExtent	domain	range
nice^z	ticks^z	tickFormat^z
copy	quantiles^l	

Ordinal [<scale>]

Sub-types: ordinal^o, band^b, point^p:

domain	rangeRound^{b,p}	padding
range	round^{b,p}	align^{b,p}
copy	paddingInner^b	bandwidth^{b,p}
unknown^o	paddingOuter^b	step^{b,p}

Axes [<axis>]

Unlike scales, axes are side-effect-having, DOM-manipulating "layouts". As such, d3-level calls must differentiate by location on page: `d3.axisTop`$_L$, `d3.axisRight`$_L$, `d3.axisBottom`$_L$,, *and* `d3.axisLeft`$_L$*. Axis methods are:*

tickValues	tickFormat	tickSize
tickSizeInner	tickSizeOuter	tickPadding

4 Shapes

Each of the following contain one eponymous top-level function that produces a "generator" which is a late-binding function that creates the indicated shape when called. Input to each generator is a data array. Output shapes are coded as "path calls," which are either svg or html canvas commands (see here) depending on the passed `<shape>.context()`*.*

Paths [<path>$_G$]

Generates: a serialized set of accumulated SVG or HTML Canvas-like path instructions.

moveTo	closePath	lineTo
quadraticCurveTo	arc	arcTo
bezierCurveTo	rect	toString

Arcs [<arc>$_G$]

Generates: circular or annular sectors for use in pie or donut charts, respectively. Here, input data must provide start and end angles.

centroid	innerRadius	outerRadius
cornerRadius	startAngle	endAngle
padAngle	padRadius	context

Lines [<line>$_G$]

Generates: a spline (smoothed curve) or polyline (piecewise-connected line) for use in a line or edge-bundling chart. There are two top-level calls: `d3.line`^{l}*, and* `d3.line-radial`^{r}*.*

x$_l$	defined	angle$_r$
y$_l$	curve	radius$_r$
	context	

Areas [<area>$_G$]

Generates an area, for use in area or difference charts. There are two top-level calls: `d3.area`^{a} *and* `d3.areaRadial`^{r}*.*

x[a] context radius[r]
x0[a] lineX0[a] innerRadius[r]
x1[a] lineY0[a] outerRadius[r]
y[a] lineX1[a] lineStartAngle[r]
y0[a] lineY1[a] lineInnerRadius[r]
y1[a] angle[r] lineEndAngle[r]
defined startAngle[r] lineOuterRadius[r]
curve endAngle[r]

Curves **[d3.curve-F]**

These are not shapes, but passed to lines & areas under their `<shape>.curve()` *call.* `curves` *are algorithms that, given input data arrays ("control points") yield smooth splines.*

Basis CatmullRomOpen
BasisClosed Linear
BasisOpen LinearClosed
Bundle MonotoneX
Cardinal MonotoneY
CardinalClosed Natural
CardinalOpen Step
CatmullRom StepAfter
CatmullRomClosed StepBefore

One can also create custom curves.

Links **[<link>G]**

Generate a smooth line segment between passed source and target points for use in tree diagrams.

∃ *vertical*[l]*,* *horizontal*[l]*, and radial*[r] *links.*

source x[l] context
target y[l] angle[r] radius[r]

Symbols **[d3.symbol-G]**

Generate a `symbol`*:*

Circle Star Diamond
Cross Square Triangle
Wye

Symbol generators provide the following methods:

item size context

Polygons **[d3.polygon-G]**

`d3.polygonHull` *builds a polygon that covers an array of input points. Other top level functions access properties of the resulting polygon:*

Area Centroid Contains Length

5 Colors

Color Creation **[d3]**

Create a `color` *from a* `<color_spec>`*:*

rgb lab lch
hsl hcl cubehelix

...where `<color_spec>` *is a string that can be the name of the color or a type-specific constructor:*

`rgb(255, 255, 255)` % RGB
`hsl(120, 50%, 20%)` % HSL
`#ffeeaa` % hex

Color Properties **[<color>]**

Get `color` *properties, or yield an externally-usable* `<color_spec>` *string:*

opacity copy formatHex
rgb displayable formatHsl
toString formatRgb

Derivative Colors **[<color>]**

Return a new, derivative `color`*:*

brighter darker

Color Schemes **[d3]**

Categorical schemes, prefixed with scheme-

Category10 Dark2 Pastel1
Accent Paired Pastel2
Set1 Set2 Set3
Tableau

Diverging schemes, prefixed with interpolate- (for continuous) or scheme- (for discrete):

BrBG PRGn RdBu RdYlBu
PiYG PuOr RdGy RdYlGn
Spectral

Sequential, single-hue schemes, prefixed with interpolate (continuous) or scheme- (discrete):

Blues Greens Greys Oranges
Purples Reds

Sequential, multi-hue schemes, prefixed with interpolate- or scheme-:

BuGn BuPu GnBu OrRd
PuBu PuBuGn PuRd RdPu
YlGn YlGnBu YlOrBr YlOrRd

Sequential, multi-hue schemes, available only in continuous form, prefixed with interpolate-:

Cividis Cool CubehelixDefault
Inferno Magma Plasma
Turbo Viridis Warm

Cyclical schemes, prefixed with interpolate-:

Rainbow Sinebow

6 Interactions

Dragging **[<drag>]**
Brushing **[<brush>]**
Zooms **[<zoom>]**

7 Transitions & Animation

General Pattern

Easings

Visually "ease" the rate (acceleration) at which objects change their velocity. Preface the following with `ease`*, and optionally suffix with one of:* `In`*,* `Out`*, or* `InOut`*:*

Linear Quad Cubic
Sin Exp Circle
Elastic Back Bounce

Interpolators **[d3]**

An `interpolator` *is a function that takes a number* $t \in [0,1]$ *and yields intermediary values in the domain-space of the specific interpolator. The following functions take two parameters that bookend the interpolation range (except for* `-Discrete`*,* `-Basis`*, and* `-BasisClosed`*, which take a single array), and yield interpolators (for color interpolators, see "color" section):*

interpolate -Round -String
-Date -Array -Numb. Array
-Object -TransformCss -Svg
-Zoom -Discrete -Basis
-BasisClosed

The last two are "splines", which produce non-linear interpolators roughly following the given array. In addition, `d3.piecewise` *generates a piecewise interpolation visiting the* n *points of its input array, and* `d3.quantize` *generates* n *samples of a passed interpolator.*

Timers

8 Layouts

Chord Layout[L] **[<chord>]**
Force Layout[L] **[<simulation>]**
Voronoi Layout[L] **[<delaunay>]**
Pies **[<pie>]**

Lays out a set of arc angles (wedges) for use as input to `arcs`*, in creating a pie chart.*

value sortValues endAngle
sort startAngle padAngle

Sankey Layout[L] **[d3]**
Pack Layout[L] **[d3]**
Partition Layout[L] **[d3]**
Cluster Layout[L] **[d3]**

Used to create a dendogram diagram.

Treemap Layout[L] **[d3]**
Stacks[L] **[<stack>]**

Generates stacking positions (in a multidimensional array $m \times n$ *for* m *series,* n *points), which are used as input to* `areas` *in creating steamgraphs, or directly in positioning stacked bars.*

keys value offset
order

Top-level algs to pass to order, offset methods:

order offset
-Order -Expand
-Ascending -Diverging
-Descending -None
-InsideOut -Silouhette
-None -Wiggle
-Reverse

9 Geography

Paths
Projections
Spherical Math
Spherical Shapes
Streams
Transforms

10 Miscellaneous

Quadtrees
Random Numbers