D3 3.0, released December 2012, is the first major release since 2.0 was released in August 2011. Since 2.0.0, there have been 10 minor releases and 37 patch releases. 3.0 includes more than 400 commits, significant new features and improvements. In accordance with semantic versioning, this rare major release also includes several backwards incompatibilities. Major releases are needed to keep the API and the code lean by removing deprecated, broken or confusing functionality. This document guides you on how to upgrade from 2.x to 3.0.

How to Upgrade

If you're using the official hosted copy of D3, simply **replace d3.v2.min.js with d3.v3.min.js** in your script tag, like this:

```
<script src="//d3js.org/d3.v3.min.js" charset="utf-8"></script>
```

(There's also a d3.v3.js for development.) Also, check that you have the DOCTYPE and charset set correctly on your HTML page:

```
<!DOCTYPE html>
<meta charset="utf-8">
```

If you'd prefer to host your own copy of D3, download the zipball or clone the repository and pull out the contained d3.js and d3.min.js. Don't copy-and-paste the JavaScript contents from your browser; that can corrupt UTF-8 characters.

Selections

The deprecated **selection.map method has been removed**; use selection.datum instead.

Transitions

D3's transition subsystem has been significantly overhauled for 3.0 to make it easier to construct complex sequences of transitions. If you're using transitions extensively, I recommend also reading Working with Transitions.

The first change is that **transition.attr, transition.style and transition.text now evaluate their property functions immediately**. In 2.x, these functions were evaluated asynchronously when the transition started, which was frequently confusing! Consider the following code:

```
// First transition the line to the new data.
line.datum(newData).transition().attr("d", line);

// Then transition the y-axis.
y.domain(newDomain);
line.transition().delay(250).attr("d", line);
```

You might expect this code to first transition the line to the new data, and then transition to the new domain. In 2.x, however, this would not work because the first transition.attr would be evaluated *after* the y-scale's domain changes. While deferred evaluation is occasionally what you want, immediate evaluation is much easier to understand and debug, so that's what transitions do in 3.0. (The selection.attr, selection.style, and related methods have always used immediate evaluation for this reason.) This means you can now easily specify transitions that depend on external

state, as in the above example showing chained transitions of data and axes. You can also create transitions within for loops without worrying about the dreaded closures in loops problem.

The other big change is that **transition.select and transition.selectAll now reselect existing transitions** rather than creating new transitions. This means that you can schedule a transition on a set of elements—say an axis—and then reselect a subset of those elements to customize the transition. This technique of customizing axes is called *postselection*, and in 3.0 you can use it for transitions as well as selections. For example, if you want to override the text-anchor for axis labels:

```
svg.select(".x.axis").transition()
    .call(xAxis)
  .selectAll("text")
    .style("text-anchor", "start");
```

In 2.x, transition.select and transition.selectAll would create a new transition that would conflict with the existing transition. Like selections, transitions in 3.0 and now stored entirely in the DOM, and thus can be reselected. Related to this, **transition.transition now creates a new transition that is scheduled to start when the originating transition ends**. This makes it very easy to create chained transitions, say from stacked to grouped bars, without the hassle of listening for "end" events.

```
rect.transition()
    .duration(500)
    .delay(function(d, i) { return i * 10; })
    .attr("x", function(d, i, j) { return x(d.x) + x.rangeBand() / n * j; })
    .attr("width", x.rangeBand() / n)
  .transition()
    .attr("y", function(d) { return y(d.y); })
    .attr("height", function(d) { return height - y(d.y); });
```

The rarely-used **d3.tween method has been removed**. This previously provided a way to override the interpolator used during a transition. Use transition.attrTween, transition.styleTween or transition.tween instead.

Requests

If you load external data via d3.xhr, **consider changing your callback to take an `error` as the first argument**. While the old single-argument callback is still supported, it is now deprecated in favor of the two-argument version.

If an error occurs loading the resource, you can use the error argument to diagnose the problem, to retry, or to inform the user. Examples of errors include network issues (such as being offline), or missing files (404) or unavailable servers (503). It also means you can now distinguish between a successfully-loaded JSON file that contains `null` and an error.

In 2.x, you might have written this:

```
d3.json("my-data.json", function(data) {
  console.log("there are " + data.length + " elements in my dataset");
});
```

While the above code still works, the recommended code in 3.0 handles errors, too:

```
d3.json("my-data.json", function(error, data) {
  if (error) return console.log("there was an error loading the data: " + error);
  console.log("there are " + data.length + " elements in my dataset");
});
```

By adopting the standard {error, result} callback convention established by Node.js, D3 can now be used with standard helpers for asynchronous JavaScript, such as Queue.js. For example, if you wanted to load multiple resources in 2.x, you probably would have loaded them serially by nesting callbacks:

```
d3.json("us-states.json", function(states) {
  d3.tsv("us-state-populations.tsv", function(statePopulations) {
    // display map here
  });
});
```

Loading resources serially is slow because it doesn't use the user's network connection efficiently. (Also, if one of the above requests fails, subsequent serialized requests will still continue because the error is ignored!) It's better to parallelize those requests. You can do that manually in vanilla JavaScript, but it's a pain since you need to track the number of outstanding requests to determine when all resources are available; it's much easier to use Queue.js:

```
queue()
    .defer(d3.json, "us-states.json")
    .defer(d3.tsv, "us-state-populations.tsv")
    .await(ready);

function ready(error, states, statePopulations) {
  // display map here
}
```

There are a number of other improvements to d3.xhr, such as the ability to listen for progress events and set request headers. For example:

```
var xhr = d3.json(url)
    .on("progress", function() { console.log("progress", d3.event.loaded); })
    .on("load", function(json) { console.log("success!", json); })
    .on("error", function(error) { console.log("failure!", error); })
    .get();
```

See the API reference for details.

Geo

D3 3.0 includes a powerful new geographic projection system featuring three-axis rotation, antimeridian cutting and adaptive resampling. (And there's also TopoJSON for more efficient representation of geometry.) These changes are almost entirely backwards-compatible.

One gotcha is that **d3.geo.path now observes the right-hand rule for polygons**. Geographic features are defined in spherical coordinates. Thus, given a small polygon that approximates a circle, we might assume that this polygon represents an island. However, an equally valid interpretation is that this polygon represents everything *but* the

island; that is, the polygon of the sea surrounding the island. (See Jason's geographic clipping examples for more.) In 2.x, it was not possible to represent polygons that were larger than a hemisphere. By applying the right-hand rule, sub-hemisphere polygons in 3.0 must have clockwise winding order. If your GeoJSON input has polygons in the wrong winding order, you must reverse them, say via ST_ForceRHR; you can also convert your GeoJSON to TopoJSON, and this will happen automatically.

Many new projections are available in the d3.geo.projection plugin. Correspondingly, the **rarely-used Bonne projection has been moved** to a plugin, and the modal **d3.geo.azimuthal projection has been replaced** with separate projections for each mode: d3.geo.orthographic, d3.geo.azimuthalEqualArea, d3.geo.azimuthalEquidistant, d3.geo.stereographic and d3.geo.gnomonic. The **albers.origin method has also been replaced** by projection.rotate and projection.center. Lastly, the alias **d3.geo.greatCircle has been removed**; use the identical d3.geo.circle instead. Also, did you know that you can now use d3.geo.circle to draw circles? This is an easy way to approximate Tissot's indicatrix.

Arrays

The rarely-used **d3.first and d3.last methods have been removed**; in most cases you want to use d3.min or d3.max. If for some reason you want the minimum element rather than the minimum value, you can simply use array.reduce:

```
var first = objects.reduce(function(p, v) { return p.value < v.value ? p : v; });
```

If you want a selection algorithm (not to be confused with a D3 selection) to select the top or bottom K of an ordered set, consider using Crossfilter's heapselect implementation. The **d3.split helper has also been removed**, since d3.svg.line and d3.svg.area now provide a defined property for handling missing data.

Geom

The rarely-used **d3.geom.contour method has been moved to a plugin**. The **d3.geom.quadtree method no longer supports array input**, where points are specified as [x, y]; instead, you should specify points as objects {x: x, y: y}.

Layouts

Hierarchy layouts no longer support wrapping input data to create nodes. Instead, the layout assigns the computed properties value, depth, etc. on the input data directly. If you were invoking the hierarchy layout using hierarchy.nodes, rather than using the deprecated method of invoking the layout directly, then you shouldn't be affected by this change. All of the official examples were changed to use hierarchy.nodes ages ago.

SVG

The aliases **d3.svg.mouse and d3.svg.touches have been removed**; use d3.mouse and d3.touches instead. These methods are identical, and were moved in an earlier minor release that added support for HTML as well as SVG elements.

Other Miscellany

The main library is now called `d3.js` in the repo, but still `d3.v3.js` on d3js.org. The official examples are now hosted on bl.ocks.org, rather than the git repo, so that they are easier to find and fork.

UglifyJS2 is now used for minification; run `npm install` to ensure you have the latest version.