Graphs in XForms

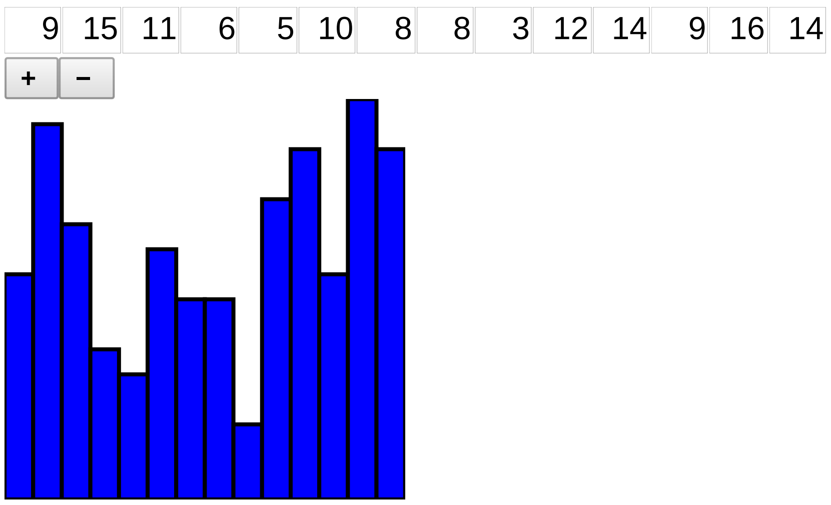
Contents

Introduction

I have some data:

```
<y>9</y><y>15</y><y>11</y><y>6</y><y>5</y><y>10</y><y>8</y>
  <y>8</y><y>3</y><y>12</y><y>14</y><y>9</y><y>16</y><y>14</y>
```

and I want to see that data as a histogram. Something like this:



Source

SVG

The histogram is displayed using SVG, driven from the data in the XForms. It is a series of rectangles, one per data point, of equal width, and of a height that depends on the value of the datapoint.

We load the data:

```
<instance id="data" src="data.xml"/>
```

and bind how we get to the data values that we want:

```
<bind id="values" ref="instance('data')/y"/>
```

We are going to create values to drive the histogram, so we will collect them in a new instance:

```
<instance id="hist">
   <data xmlns="">...</data>
</instance>
```

Horizontal Space

The first value we need is how many data values there are:

```
<bind ref="n" calculate="count(bind('values'))"/>
```

(This is XForms 2.0; with earlier versions, you have to expand the bind:)

```
<bind ref="n" calculate="count(instance('data')/y)"/>
```

We are going to allocate space for the histogram, that is 100 × 100.

"*100 what?*" you ask? It doesn't matter, SVG lets you scale (it's what the S stands for). Think of the 100 as 100%.

Since there are *n* values, we will split the horizontal space into n vertical bars, and we will share the horizontal space amongst them by giving them a width of 100 ÷ n units each:

```
<bind ref="width" type="double" calculate="100 div ../n"/>
```

Vertical Space

The vertical space is distributed over the values. If they are all positive, then the space will be distributed over 0 to max. If they are all negative, the space will be distributed over 0 to min. And if there are negative and positive values, the space will be distributed from min to max:

```
<bind ref="min" calculate="min(bind('values'))"/>
<bind ref="max" calculate="max(bind('values'))"/>
```

That's easy. Now the range.

```
<bind ref="rmin" calculate="if(../min &lt; 0, ../min, 0)"/>
<bind ref="rmax" calculate="if(../max &gt; 0, ../max, 0)"/>
<bind ref="range" calculate="../rmax - ../rmin"/>
```

This is the range that the vertical height is distributed over:

```
<bind ref="vscale" type="double" calculate="100 div ../range"/>
```

So

- if max is 3 and min is -2, range is 5;
- if max is 3 and min is 1, range is 3;
- if max is -1 and min is -4, range is 4.

Values

9	15	11	6	5	10	8	8	3	12	14	9	16	14

+ −

n:
14
min:
3
max:
16
width:
7.142857
rmin:
0
rmax:
16
range:
16
vscale:
6.25
vscale * rmax:
100

Source

Edge cases

"But what if there are no values? What if all the values are zero?"

If there is no data then indeed several of these values get odd values. "NaN" is "Not a Number", and "Infinity" is, well, infinity.

But if there are no values, the histogram will be empty, and the values won't get used

If all the data points are zero, most values will also be zero, except the vertical scale which will be infinite, and we will end up trying to draw boxes of height zero, scaled by infinity, which will be NaN high.

So to catch that case:

```
<bind ref="vscale" type="double"
      calculate="if(../range=0, 1, 100 div ../range)"/>
```

(It doesn't matter what value we use, since zero times anything is still zero.)

Edge cases

There's one other case it would be good to catch: if any of the data values is empty, or not a number for any other reason, then `max` and `min` return `NaN` (try it on the values above). We fix that by changing the calculation for them to only select those values that are numbers:

```
<bind ref="min" calculate="min(bind('values')[number(.)!='NaN'])"/>
<bind ref="max" calculate="max(bind('values')[number(.)!='NaN'])"/>
```

OK. Now we have enough to be able to draw the histogram.

The Histogram

As was said, we're going to draw a series of rectangles, one for each data value, each of the same width, and of a height depending on the value itself. Roughly speaking like this:

```
<svg ...some svg attributes here...>
   <xf:repeat bind="values">
      <rect width="{instance('hist')/width}"
            height="{...}"
            x="{...}"
            y="{...}"
            />
   </xf:repeat>
</svg>
```

We would like to say:

```
height="{instance('hist')/vscale * .}"
```

but alas, for some unfathomable reason, SVG does not permit negative heights, so we have to do a bit more work:

```
height="{instance('hist')/vscale * if(. &lt; 0, 0 - ., .)}"
```

About SVG

Before telling you how to calculate x and y, you have to know something about how SVG works:

1. The point (0,0) is by default at the top left corner.
2. The positive horizontal direction is to the right, negative to the left.
3. The positive vertical direction is downwards, and negative upwards.

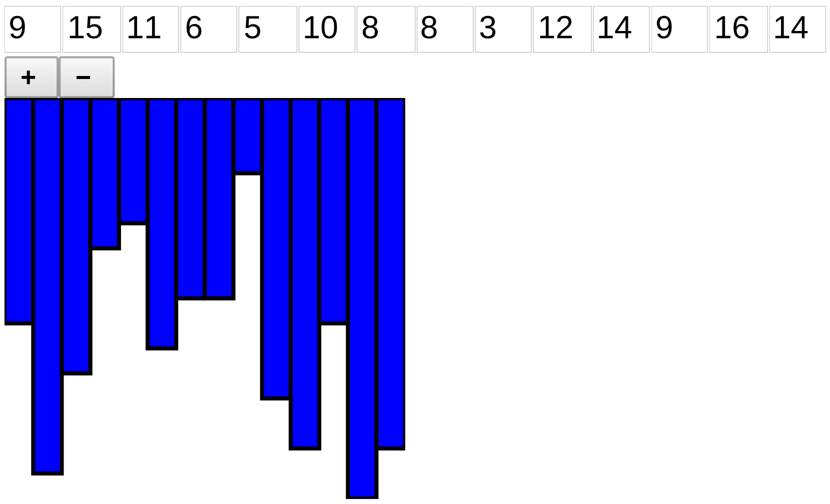
Horizontal Position

So, x is easy to calculate. The first rectangle will be at position 0; the next at 1×width; the next at 2×width, and so on.

Each item in a `repeat` has a position from 1 to n, so we have to subtract 1 to get 0 to n-1:

```
x="{(position()-1)*instance('hist')/width}"
```

So now we have a row of rectangles, of the correct height, of the correct width, at the correct horizontal position:


9 15 11 6 5 10 8 8 3 12 14 9 16 14

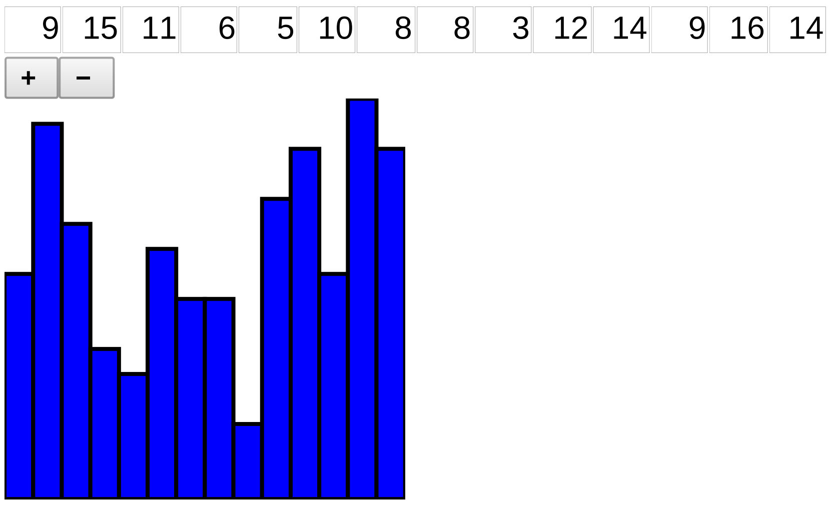
Source

Vertical Position

The base of the tallest (positive) bar is the zero line. All positive values should line up with their bases there; therefore we need to add a little to their x value to push them down. That amount is `rmax` less the value of the datapoint. Negative values need to start at that baseline, so their y value is just `rmax`. (I should point out that if SVG had allowed negative heights, both calculations would have been the same; that's the power of generalisation...)

```
y="{instance('hist')/vscale *
         if(. &lt; 0, instance('hist')/rmax,
                      instance('hist')/rmax - .) }"
```

Giving the final result:


9 15 11 6 5 10 8 8 3 12 14 9 16 14

Source

Turning it into a Graph

There you have it, a histogram with XForms. About 25 lines of XForms, depending on exactly what you count.

But *actually*, now that we have all those values at our disposal, it doesn't take much to turn the histogram into a graph.

This time, instead of drawing a box for each value, we will draw a line from one value, to the next.

Graph

So we will repeat over all values *except the last* (which has no next):

```
<xf:repeat ref="bind('values')[position() != last()]">
```

and within the repeat draw a line from that value to the next:

```
<line x1="{(position() - 1) * instance('hist')/width}"
      x2="{(position()    ) * instance('hist')/width}"
      y1="{instance('hist')/vscale * (instance('hist')/rmax - .)}"
      y2="{instance('hist')/vscale * (instance('hist')/rmax - following-sibl
      class="line"/>
```

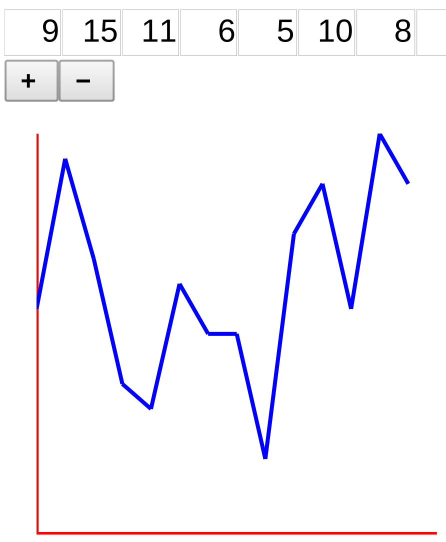
Axes

Draw (outside of the repeat of course) two axes:

```
<line x1="0"   y1="{instance('hist')/rmax * instance('hist')/vscale}"
      x2="100" y2="{instance('hist')/rmax * instance('hist')/vscale}"
      class="axis"/>
<line x1="0" y1="0"
      x2="0" y2="100"
      class="axis"/>
```

Result

And it looks like this:



9 15 11 6 5 10 8 8 3 12 14 9 16 14

Source

Width

If you look very closely (or delete all but two or three of the numbers), you will see that there is some extra space at the right-hand side.

That is because we are now graphing over one less value, so the horizontal space is divided over too many values.

That can be fixed by changing the calculation for `width` to:

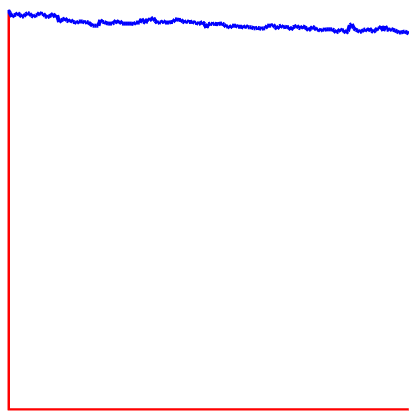
```
<bind ref="width"  calculate="100 div (../n - 1)"/>
```

Bigger data

If we graph data where the values are further from zero, with small differences between the values, we get something like this:

76.5	75.9	75.2	75.5	75.6	75.9	75.6	75.2	75.3	75.7	75.9	75.8	75.4	75.2	75.5
75.5	76	75.7	75.8	75.3	75.1	75.3	75.5	75.8	75.1	75.5	74.2	74.5	74.7	74.9
74.5	74.3	74.6	74.2	74.2	74	74.3	74.3	74.3	74.1	74.2	74.1	73.8	73.5	73.5
73.5	73.5	74.6	74.2	74.2	74	73.9	73.9	73.9	74.1	74.5	74	74.4	73.9	73.9
73.9	73.9	73.9	73.9	73.8	74.1	74.1	74.1	74.8	74.3	74	74.8	74.5	74.8	75
74.4	74	74.2	74.1	74.4	74.1	74.1	74.1	74.2	74.2	74.7	74.7	74.6	74.6	74.3
74.1	74.4	74.2	74.2	74.3	74	74.1	73.8	74	74.2	73.8	73.1	73.8	73.9	73.8
73.9	73.8	73.8	73.9	73.9	73.7	73.3	73.4	73.1	73.5	73.5	73.5	73.3	73.3	73.3
73.1	73.5	73.2	73.2	73.2	73	73	73.1	73	72.9	73	73	73.5	73.5	73.5
73.7	73.2	72.9	73.4	73.5	73.2	73.3	73.2	73.2	72.7	73.2	73.3	73.5	73.1	73
73.4	73.2	73	72.6	72.9	73.3	73.1	72.7	72.8	72.5	72.8	72.7	72.8	72.8	72.8
72.8	72.6	72.2	72.4	72.8	72.6	72.6	72.1	72.4	73.6	73.8	73.0	72.7	72.6	72.3
72.4	72.7	72.8	72.5	73	72.5	72.3	72.6	73	72.9	73.4	72.5	73.4	72.8	72.6
72.9	72.7	72.6	72.4	72.2	72.2	72.3	72.4	72.2	72.0					

+ −



Source

Avoiding zero

In such cases, we may prefer to graph over the range of actual values, if the difference between values is what interests us.

It's a simple change: we just change the definition of `rmin` and `rmax`, which define the range of values we are drawing over:

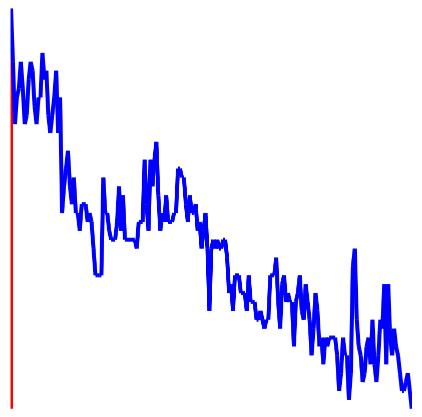
```
<bind ref="rmin"   calculate="../min"/>
<bind ref="rmax"   calculate="../max"/>
```

Result

which gives us, for the same data:

76.5	75.9	75.2	75.5	75.6	75.9	75.6	75.2	75.3	75.7	75.9	75.8	75.4	75.2	75.5
75.5	76	75.7	75.8	75.3	75.1	75.3	75.5	75.8	75.1	75.5	74.2	74.5	74.7	74.9
74.5	74.3	74.6	74.2	74.2	74	74.3	74.3	74.3	74.1	74.2	74.1	73.8	73.5	73.5
73.5	73.5	74.6	74.2	74.2	74	73.9	73.9	73.9	74.1	74.5	74	74.4	73.9	73.9
73.9	73.9	73.9	73.9	73.8	74.1	74.1	74.1	74.8	74.3	74	74.8	74.5	74.8	75
74.4	74	74.2	74.1	74.4	74.1	74.1	74.1	74.2	74.2	74.7	74.7	74.6	74.6	74.3
74.1	74.4	74.2	74.2	74.3	74	74.1	73.8	74	74.2	73.8	73.1	73.8	73.9	73.8
73.9	73.8	73.8	73.9	73.9	73.7	73.3	73.4	73.1	73.5	73.5	73.5	73.3	73.3	73.3
73.1	73.5	73.2	73.2	73.2	73	73	73.1	73	72.9	73	73	73.5	73.5	73.5
73.7	73.2	72.9	73.4	73.5	73.2	73.3	73.2	73.2	72.7	73.2	73.3	73.5	73.1	73
73.4	73.2	73	72.6	72.9	73.3	73.1	72.7	72.8	72.5	72.8	72.7	72.8	72.8	72.8
72.8	72.6	72.2	72.4	72.8	72.6	72.6	72.1	72.4	73.6	73.8	73.0	72.7	72.6	72.3
72.4	72.7	72.8	72.5	73	72.5	72.3	72.6	73	72.9	73.4	72.5	73.4	72.8	72.6
72.9	72.7	72.6	72.4	72.2	72.2	72.3	72.4	72.2	72.0					

+ −



which reveals much more clearly that the values are descending. Source

Making it a choice

Of course, it doesn't have to be either/or; it can be made a choice in the code whether to include zero in the display. Add a new value to the display data:

```
<include0>true</include0>
  ...
<bind ref="include0" type="boolean"/>
```

Change how `rmin` and `rmax` are calculated:

```
<bind ref="rmin"     calculate="if(../min < 0 or ../include0=false(), ../min
<bind ref="rmax"     calculate="if(../max > 0 or ../include0=false(), ../max
```

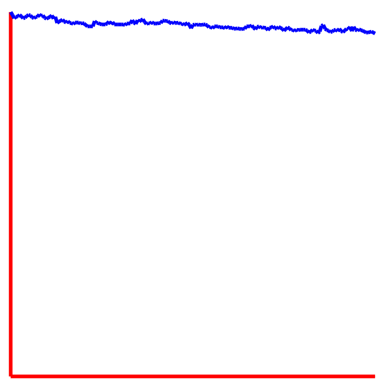
And add an input to change whether you want zero to be included or not:

```
<input ref="instance('hist')/include0"><label>zero</label></input>
```

Result

76.5	75.9	75.2	75.5	75.6	75.9	75.6	75.2	75.3	75.7	75.9	75.8	75.4	75.2	75.5
75.5	76	75.7	75.8	75.3	75.1	75.3	75.5	75.8	75.1	75.5	74.2	74.5	74.7	74.9
74.5	74.3	74.6	74.2	74.2	74	74.3	74.3	74.3	74.1	74.2	74.1	73.8	73.5	73.5
73.5	73.5	74.6	74.2	74.2	74	73.9	73.9	73.9	74.1	74.5	74	74.4	73.9	73.9
73.9	73.9	73.9	73.9	73.8	74.1	74.1	74.1	74.8	74.3	74	74.8	74.5	74.8	75
74.4	74	74.2	74.1	74.4	74.1	74.1	74.1	74.2	74.2	74.7	74.7	74.6	74.6	74.3
74.1	74.4	74.2	74.2	74.3	74	74.1	73.8	74	74.2	73.8	73.1	73.8	73.9	73.8
73.9	73.8	73.8	73.9	73.9	73.7	73.3	73.4	73.1	73.5	73.5	73.5	73.3	73.3	73.3
73.1	73.5	73.2	73.2	73.2	73	73	73.1	73	72.9	73	73	73.5	73.5	73.5
73.7	73.2	72.9	73.4	73.5	73.2	73.3	73.2	73.2	72.7	73.2	73.3	73.5	73.1	73
73.4	73.2	73	72.6	72.9	73.3	73.1	72.7	72.8	72.5	72.8	72.7	72.8	72.8	72.8
72.8	72.6	72.2	72.4	72.8	72.6	72.6	72.1	72.4	73.6	73.8	73.0	72.7	72.6	72.3
72.4	72.7	72.8	72.5	73	72.5	72.3	72.6	73	72.9	73.4	72.5	73.4	72.8	72.6
72.9	72.7	72.6	72.4	72.2	72.2	72.3	72.4	72.2	72.0					

+ −



zero

✓

Source