A Clock in XForms

Steven Pemberton, CWI Amsterdam

Version: 2019-04-02.

Introduction

As the name suggests, XForms was originally designed for dealing with forms. However, thanks to its generalised design it is suitable for much more.

One of the advantages of XML is the ability to combine markup from different vocabularies in the same document. In this example, we show how to combine SVG and XForms, in this case to display a clock, though in later articles we'll use SVG to visualise data.

Time

You can get the time in XForms with the function `local-dateTime`, which gives you the current date, time, and timezone. We'll create an instance, initialise a value, and display it:

```
<model>
   <instance>
      <clock xmlns="">
         <time/>
      </clock>
   </instance>

   <bind ref="time" calculate="local-dateTime()"/>
</model>
 ...
<output ref="time" label="time"/>
```

Which looks like this:

time
2019-12-10T13:51:19+01:00

Source

Which, if all is well, and you are looking at the live version of this document, displays your current date and time, and the difference of your local time from UTC.

You'll see that the hour part begins at position 12, and is 2 characters long. You can extract this with the function `substring(local-dateTime(), 12, 2)`.

So, let's add some values to the instance, initialise them, and add them to the output:

```
<instance>
   <clock xmlns="">
      <time/>
      <h/><m/><s/>
   </clock>
</instance>

<bind ref="time" calculate="local-dateTime()"/>
<bind ref="h" calculate="substring(../time, 12, 2)"/>
<bind ref="m" calculate="substring(../time, 15, 2)"/>
<bind ref="s" calculate="substring(../time, 18, 2)"/>
 ...
```

```
<output ref="time" label="time"/>
<output value="concat(h, ':', m, ':', s)"/>
```

Which gives this:

time
2019-12-10T13:51:19+01:00
13:51:19

Source

So far so good. But we want a ticking clock. At start-up we catch the `xforms-ready` event, and respond by dispatching an event we shall call `tick`:

```
<action ev:event="xforms-ready">
   <dispatch name="tick" targetid="m"/>
</action>
```

(we've also added an `id="m"` to the `model` element).

We catch the `tick` event in a similar way, and respond by resetting the `time` value, and then dispatch another `tick`, but after a 1 second delay (expressed in milliseconds):

```
<action ev:event="tick">
   <setvalue ref="time" value="local-dateTime()"/>
   <dispatch name="tick" targetid="m" delay="1000"/>
</action>
```

This is the only change. And now we have a ticking clock:

time
2019-12-10T13:51:23+01:00
13:51:23

Source

(Actually, we can make one more change, since the `<bind ref="time" .../>` is no longer needed.)

But we really want an analogue-style clock. So we add three values to the instance for the hands that will hold the angle that the hand has when displayed, as well as its length, and class:

```
<instance>
   <clock xmlns="">
      <time/>
      <h/><m/><s/>
      <hand length="30" class="h"/>
      <hand length="40" class="m"/>
      <hand length="40" class="s"/>
   </clock>
</instance>
```

It doesn't matter what units the lengths are expressed in, because the clock will be scalable, but think of the number as a percentage in terms of the whole clock.

Now to calculate the angles. There are 360° in a circle, and the second hand goes round once every 60 seconds. So each second it moves (360÷60)° = 6°:

```
<bind ref="hand[@class='s']" calculate="../s * 6"/>
```

Same for the minutes:

```
<bind ref="hand[@class='m']" calculate="../m * 6"/>
```

The hour hand is slightly more complicated: although each hour represents (360÷12) degrees, which is 30 degrees, the time is expressed in the 24 hour clock; additionally the hour hand doesn't jump from one hour to another, but slowly moves between them, so we have to take the minutes into account as well.

So to get the hours in a 12 hour clock, we can say `(h mod 12)` to give us the range 0 to 11. Each 30° segment represents 60 minutes, so each minute adds an additional ½°:

```
<bind ref="hand[@class='h']" calculate="(../h mod 12)*30 + (../m div 2)"/>
```

Actually though, the mod is not necessary, because 13*30 = 390, and rotating by 390 degrees has the same effect as rotating by 30 degrees. So this is simpler:

```
<bind ref="hand[@class='h']" calculate="(../h * 30) + (../m div 2)"/>
```

Now to display our analogue-style clock. For this we use SVG (Scalable Vector Graphics):

```
<svg (some extra parameters here)>
   the clock here
</svg>
```

The first bit is the bezel around the clock, a circle:

```
<circle r="45" class="bezel"/>
```

With a radius of 45, the clock will have a width of 90%, leaving a 10% border around it.

The `class` attribute allows us to set line width and colour in the CSS:

```
.bezel { stroke-width: 1; stroke: black }
```

Now for the hands. We do a repeat over the hands to display them:

```
<xf:repeat ref="hand">
   <line x1="0" y1="0"
         x2="0" y2="-{@length}"
         transform="rotate({.})"
         class="{@class}"/>
</xf:repeat>
```

This draws a line for each hand, from `(0,0)` (which is the centre of the clock), to `(0, -length)`. The length point is negative, since negative is the upwards direction in SVG. So this draws a line from the centre to the 12 o'clock position. We transform this by rotating it by the angle we calculated earlier; and again use a bit of CSS for the width and colour:

```
.h, .m { stroke-width:4; stroke: black }
.s     { stroke-width:2; stroke: red }
```

And finally a little circle in the middle:

```
<circle r="2" class="centre"/>
```

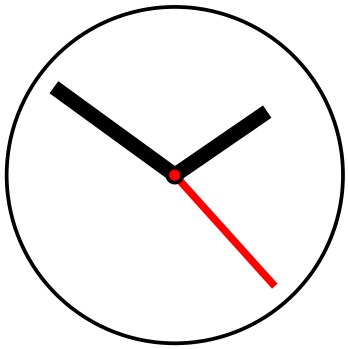
Putting all this together:

```
<svg ...>
   <circle r="45" class="bezel"/>
   <xf:repeat ref="hand">
      <line x1="0" y1="0"
            x2="0" y2="-{@length}"
            transform="rotate({.})"
```

```
            class="{@class}"/>
   </xf:repeat>
   <circle r="2" class="centre"/>
</svg>
```

which looks like this:

time
2019-12-10T13:51:23+01:00



Source

Let's add some marks around the bezel for the hours. There are twelve of them, at 30° intervals:

```
<instance>
   <clock xmlns="">
      <time/>
      <h/><m/><s/>
      <hand length="30" class="h"/>
      <hand length="40" class="m"/>
      <hand length="40" class="s"/>
      <mark/>
   </clock>
</instance>
```

Only one has been added here; we add the other eleven at start-up:

```
<action ev:event="xforms-ready">
   <insert ref="mark" while="count(mark &lt; 12)"/>
   <dispatch name="tick" targetid="m"/>
</action>
```

and we'll give them an angle, just like with the hands:

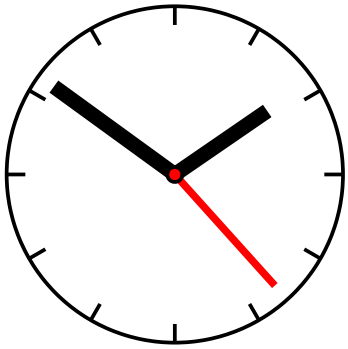
```
<bind ref="mark" calculate="count(preceding-sibling::mark) * 30"/>
```

and within the SVG, we display them with a repeat just as with the hands:

```
<xf:repeat ref="mark">
   <line x1="0" y1="-40"
         x2="0" y2="-45"
         transform="rotate({.})"
         class="mark"/>
</xf:repeat>
```

And this all looks like this:

time
2019-12-10T13:51:23+01:00



Source

Or even fancier, we'll have a mark for every minute. Add a length and class to the mark in the instance:

```
<mark length="" class=""/>
```

Add 60 of them instead of 12:

```
<action ev:event="xforms-ready">
   <insert ref="mark" while="count(mark &lt; 60)"/>
   <dispatch name="tick" targetid="m"/>
</action>
```

Change the bind for the angle:

```
<bind ref="mark" calculate="count(preceding-sibling::mark) * 6"/>
```

Add binds for the length and class, so that every 5th mark is different:

```
<bind ref="mark/@class"
      calculate="if(count(../preceding-sibling::mark) mod 5) = 0, 'hmark', 'mark')"/>
<bind ref="mark/@length"
      calculate="if(count(../preceding-sibling::mark) mod 5) = 0, 8, 4)"/>
```

And update the SVG:

```
<xf:repeat ref="mark">
   <line x1="0" y1="{@length - 45}"
         x2="0" y2="-45"
         transform="rotate({.})"
         class="{@class}"/>
</xf:repeat>
```

Giving us:

time
2019-12-10T13:51:23+01:00

Source

As you can see, using SVG to display dynamically generated values in XForms is really easy. In later articles, we will use it to visualise sets of data.