Live XML Data

Steven Pemberton

CWI, Amsterdam

Abstract

XML is often thought of in terms of documents, or data being transferred between machines, but there is an aspect of XML often overlooked, and that is as a source of live data, that can be displayed in different ways in real time, and used in interactive applications.

In this paper we talk about the use of live XML data, and give some examples of its use.

1. Introduction

In [1], Tim Bray, one of the developers of XML, said

> "You know, the people who invented XML were a bunch of publishing technology geeks, and we really thought we were doing the smart document format for the future. Little did we know that it was going to be used for syndicated news feeds and purchase orders."

In other words, they did they not anticipate XML's use outside of documents and publishing, as data, as interactive documents, and so on.

But with the increasing availability of apps, live data is becoming more and more significant.

2. Live Data

Live XML data is the use of XML in an application where the data is constantly updated, either by repeated polling of an external source, or through interaction with the user, or a combination of both.

To give an example [2], it is currently good practice to give suggestions if a user is searching in a large database or similar. Using XML and XForms [3], [4], [5], [6], [7], it is easy to specify this: the search string is kept in instance data:

```
<root xmlns="">
  <search/>
</root>
```

which is input with an *incremental* control, that updates the data each time a key is pressed:

```
<input ref="search" incremental="true">
  <label>Search: </label>
</input>
```

Whenever the value is changed in the control (which invokes an `xforms-value-changed` event), the data is submitted to the site (in this case wikipedia):

```
<send ev:event="xforms-value-changed"
      submission="s1"/>
```

The submission that causes this specifies that the results of the submission are returned into a different instance

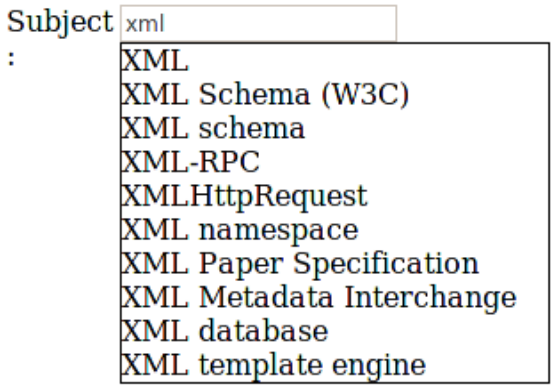
```
<submission id="s1" resource="
http://en.wikipedia.org/w/api.php?action=opensearch
" method="get"
  replace="instance"
  instance="iresults"/>
```

The results are then displayed to the user, as a series of 'triggers', which when clicked on, set the value of the search string:

```
<repeat id="results"
        nodeset="instance('iresults')/*[2]/*">
  <trigger appearance="minimal">
  <label><output value="."/></label>
    <action ev:event="DOMActivate">
      <setvalue
        ref="instance('isearch')/search"
        value="instance('iresults')
               /*[2]/*[index('results')]" />
    </action>
  </trigger>
</repeat>
```


doi:10.14337/XMLLondon14.Pemberton01

And the result looks like this:



This is a general idiom that can be used in many places: source data is changed in some way, possibly by interactions from the user, and this causes data to be updated from external sources.

3. XForms

XForms is a language originally developed for dealing with forms on the web. However, thanks to the generality of its design it was soon realised that, with a little more generality, it could be used for more general applications as well. So since XForms version 1.1, applications can be built with XForms. In fact, a form is really just the collection of data, some calculation, and some output, as well as submission of data. But this is a actually the description of an application as well. The only real noticeable difference is the manner in which the data is collected and presented.

XForms has been in use for more than a decade now, by a wide range of users including the BBC, the Dutch national weather service, NASA, and Xerox, just to name a few. Experience has shown that XForms greatly reduces the time needed to produce an application (by about an order of magnitude). This is largely due to the approach used by XForms, of declaratively specifying *what* is to be achieved, rather than *how* to achieve it.

4. An Example

In XForms you can put the URL of an image in your data:

```
<instance>
  <data xmlns="">
    <url>
      http://tile.openstreetmap.org/10/511/340.png
    </url>
  </data>
</instance>
```

and output it with

```
<output ref="url"/>
```

This would give as output:

```
http://tile.openstreetmap.org/10/511/340.png
```

But if you add a `mediatype` to the `<output>`, the image itself is output instead:

```
<output ref="url" mediatype="image/*" />
```

5. URL Structure

An Open Street Map URL is made up as:
`http://<site>/<zoom>/<x>/<y>.png`

So we can represent that in XForms data:

```
<instance>
  <map xmlns="">
    <site>http://tile.openstreetmap.org/</site>
    <zoom>10</zoom>
    <x>511</x>
    <y>340</y>
    <url/>
  </map>
</instance>
```

and calculate the URL from the parts:

```
<bind ref="url"
      calculate="concat(../site, ../zoom, '/',
                        ../x, '/', ../y, '.png')"/>
```

But now that we have the data, we can also input the different parts:

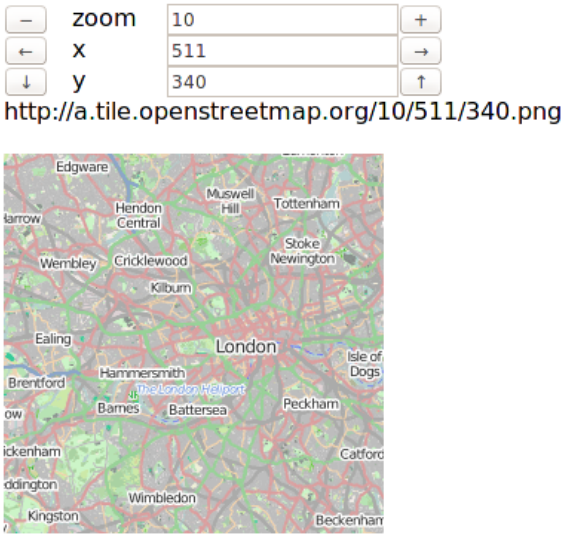
```
<input ref="zoom"><label>zoom</label></input>
```

This means that we can enter different values for the tile coordinates, and because XForms keep all relationships up-to-date, a new tile URL is calculated and the corresponding tile is displayed.

However, since entering numbers like this is inconvenient, we can also add some nudge buttons, of the form:

```
<trigger>
  <label>→</label>
  <setvalue ev:event="DOMActivate" ref="x"
            value=". + 1"/>
</trigger>
```

so it looks like this:



6. Zoom

A problem with this is that while the x and y nudge buttons work fine, the zoom button doesn't. This is because at each level of zoom the x and y coordinates change: at the outermost level of zoom, 0, there is one tile, x=0, y=0. At level 1, the coordinates double in both direction, [0-1], so there are 4 tiles; at level 2, the coordinates are [0-3], and there are 8 tiles, 16 at level 3, and in general 2z at level z (up to level 18).

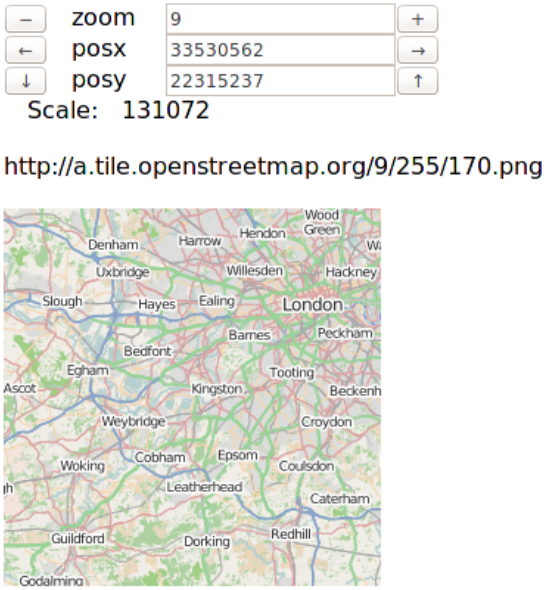
So to make zoom work properly, we must save our location in world coordinates, each value between 0 and 226 (which is the 18 levels of zoom, plus 8 bits for the 256 pixels of each tile), and then calculate the tile at any level of zoom from that:

scale=226 - zoom
x=floor(posx/scale)
y=floor(posy/scale)

In XForms:

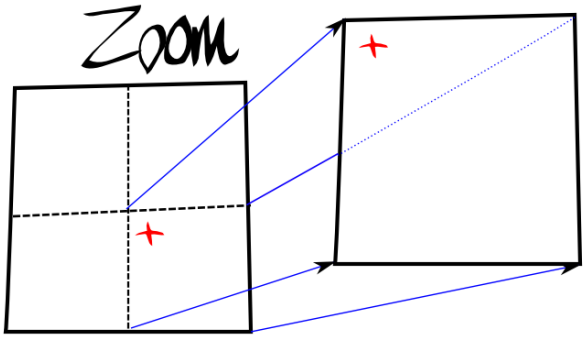
```
<bind ref="scale"
      calculate="power(2, 26 - ../zoom)"/>
<bind ref="x"
      calculate="floor(../posx div ../scale)"/>
<bind ref="y"
      calculate="floor(../posy div ../scale)"/>
```

Now when you zoom in and out, the area remains the same:

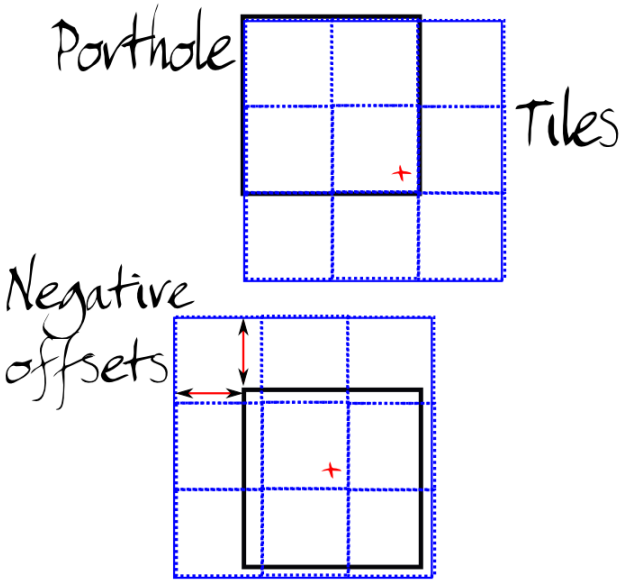


7. Location, location, location

You'll notice from the two images above that we got the tile that contains our location, but the location (in this case, central London) is at a different part of the tile. This is because if you have a tile where the location is in the middle of the tile, when you zoom in, you get one of the 4 quadrants, and so by definition, the location is no longer at the centre of the tile:

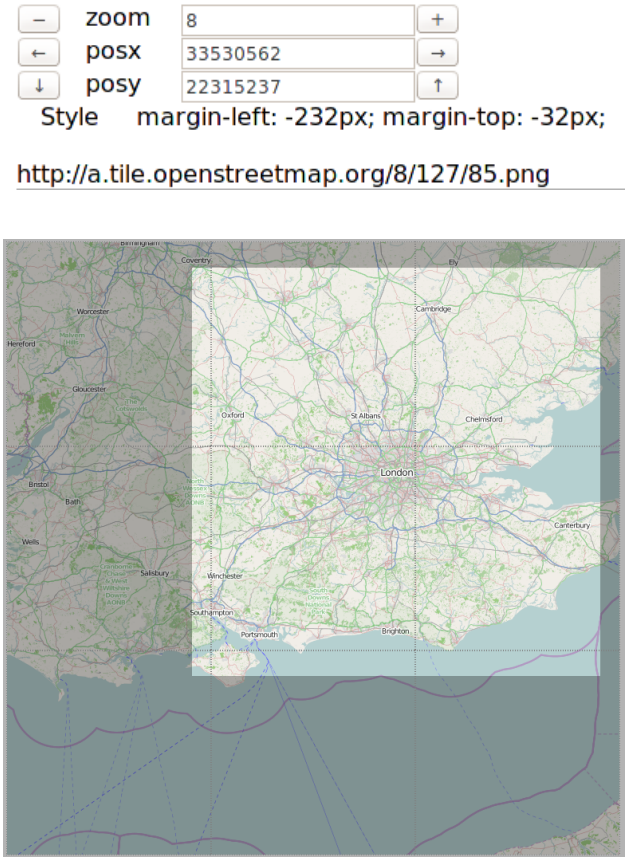


From a usability point of view of course, we want our location to remain in the middle of the view, so to achieve this, we create a 3×3 array of tiles, with a porthole over it. The porthole stays static, and we shift the tiles around underneath so that our location remains in the centre. This we do by calculating offsets that the tile array has to be shifted by, and then using these to construct a snippet of CSS to move the tile array:




```
<bind ref="offx"
      calculate="0 - floor(((
                        ../posx - ../x * ../scale)
                        div ../scale)*../tilesize)"/>
<bind ref="offy"
   calculate="0 - floor(((
                      ../posy - ../y * ../scale)
                      div ../scale)*../tilesize)"/>
 ...
<div style="margin-left: {offx};
            margin-top: {offy}">
```

Now we have a live map, where we can zoom in and out, and pan left and right and up and down. Here is a view also showing the parts that would normally not be visible, outside of the porthole:



Unfortunately, in a paper, you can't see interactive applications like this working. You can see it in action, and look at the code, at [8].

8. Mouse

Of course, what we *really* want is to be able to drag the map around with the mouse, not have to click on nudge buttons. Now we're really going to see the power of live data! We will want to know the position of the mouse, and the state of the button, up or down. So we create instance data for that:

```
<mouse>
  <x/><y/><state/>
</mouse>
```

and then we catch the mouse events:

```
<action ev:event="mousemove">
  <setvalue ref="mouse/x"
            value="event('clientX')"/>
  <setvalue ref="mouse/y"
            value="event('clientY')"/>
</action>
<action ev:event="mousedown">
  <setvalue ref="mouse/state">down</setvalue>
</action>
<action ev:event="mouseup">
  <setvalue ref="mouse/state">up</setvalue>
</action>
```

Now we have live data for the mouse!

We can show the state of the mouse by changing the mouse cursor from a hand into a clenched hand:

```
<div style="cursor: {
  if(mouse/state='up', 'pointer', 'move')
}">...
```

9. Capturing a move

The last bit is that we want is to save the start and end point of a move, so we can calculate how far we have dragged. The instance data is extended:

```
<mouse>
  <x/><y/><state/>
  <start><x/><y/></start>
  <end><x/><y/></end>
  <move><x/><y/></move>
</mouse>
```

We capture the start point of the drag when the mouse button goes down:

```
<action ev:event="mousedown">
  <setvalue ref="mouse/state">down</setvalue>
  <setvalue ref="mouse/start/x"
            value="event('clientX')"/>
  <setvalue ref="mouse/start/y"
            value="event('clientY')"/>
</action>
```

While the mouse button is down, we save the end position:

```
<bind ref="mouse/end/x"
      calculate="if(mouse/state = 'down',
                    mouse/x, .)"/>
<bind ref="mouse/end/y"
      calculate="if(mouse/state = 'down',
                    mouse/y, .)"/>
```

And calculate the distance moved as just end - start:

```
<bind ref="mouse/move/x"
      calculate="mouse/end/x - mouse/start/x"/>
<bind ref="mouse/move/y"
      calculate="mouse/end/y - mouse/start/y"/>
```

10. Dragging the map

So now we have the scaffolding we need to be able to drag the map. You may recall that the position of the map is recorded in `posx` and `posy`. That position now also depends on the mouse dragging. So we add instance data to record the last position:

```
<lastx/><lasty/>
```

and add a calculation to keep `posx` and `posy` updated (remember `scale` is the number of positions represented on a tile, so we divide by the tile size to get the number of positions represented by a pixel):

```
<bind ref="posx"
      calculate="../lastx -
                 ../mouse/move/x *
                 (../scale div ../tilesize)"/>
<bind ref="posy"
      calculate="../lasty -
                 ../mouse/move/y *
                 (../scale div ../tilesize)"/>
```

and only one other thing, namely reset `lastx` and `lasty` when the dragging stops:

```
<action ev:event="mouseup">
  <setvalue ref="lastx"
            value="posx"/>
  <setvalue ref="lasty"
            value="posy"/>
  <setvalue ref="mouse/start/x"
            value="mouse/end/x"/>
  <setvalue ref="mouse/start/y"
            value="mouse/end/y"/>
</action>
```

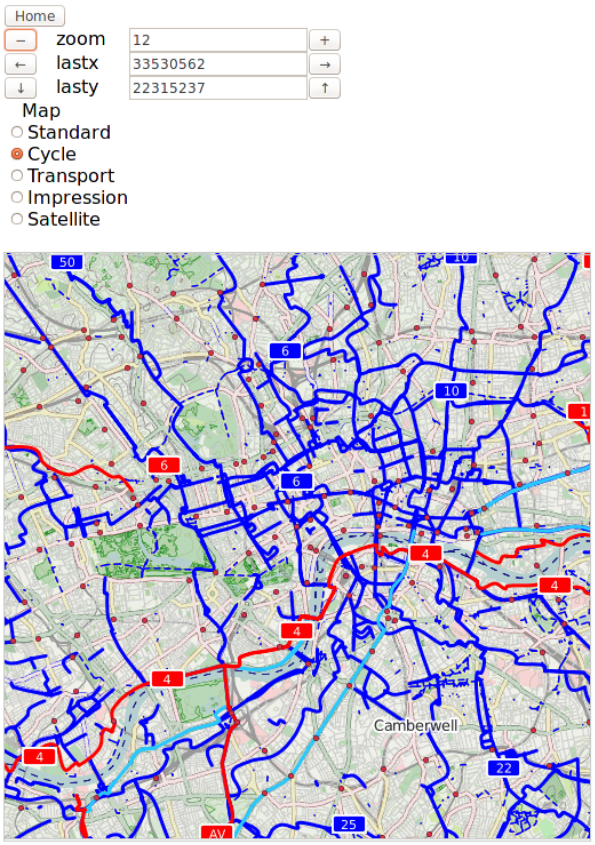
Now it is possible to drag the map around. Although from the user's point of view it feels like you are grabbing the map and dragging it around, all that is happening underneath is that we are tracking the live data representing the mouse, and using it to alter the live data that represents the centre of the map.

11. Bells. Whistles

Once we have this foundation, it is *trivial* to add things like a "Home" button, to add keystroke shortcuts, to zoom in and out with the mouse wheel, or to select tiles for another version of the map. For instance:

```
<select1 ref="site">
  <label>Map</label>
  <item>
    <label>Standard</label>
    <value>
      http://tile.openstreetmap.org/
    </value>
  </item>
  <item>
    <label>Cycle</label>
    <value>
      http://tile.opencyclemap.org/cycle/
    </value>
  </item>
  <item>
    <label>Transport</label>
    <value>
      http://tile2.opencyclemap.org/transport/
    </value>
  </item>
  ...
</select1>
```

Thanks to the live data, any time a different value is selected for "site", all the tiles get updated, without any further work from us.



12. Implementation

The implementation used in the online version of this application is XSLTForms [9], a client-side implementation of XForms that runs in all modern browsers, using a mixture of XSLT and Javascript. However, the code only uses standard XForms, and does not use any special facilities of this implementation. As long as the XForms implementation correctly catches the DOM mouse events used, the code should work in any implementation of XForms.

13. Conclusion

In a very abstract sense, a map like the one presented above can be seen as the presentation of two values, an x and y coordinate, overlaid with an input control to affect the values of x and y. The ability of XForms to abstract the data out of an application and make the data live via simple declarative invariants that keep related values up to date makes the construction of interactive applications extremely simple. The above example map application is around 150 lines of XForms code, in sharp comparison with the several thousand lines that a procedural programming language would need.

References

[1] *A Conversation with Tim Bray, Searching for ways to tame the world's vast stores of information.*. Jim Gray. ACM Queue. 20-25. 3. 1. February 2005. http://queue.acm.org/detail.cfm?id=1046941

[2] *WIKIPEDIA OpenSearch Test Form*. Alain Couthures. http://www.agencexml.com/xsltforms/wikipediasearch.xml

[3] *XForms 1.0*. Micah Dubinko, Leigh Klotz, Roland Merrick, and T V Raman. W3C. 2003. http://www.w3.org/TR/2003/REC-xforms-20031014/

[4] *XForms 1.1*. John Boyer. W3C. 2009. http://www.w3.org/TR/2009/REC-xforms-20091020/

[5] *XForms 2.0 (draft)*. John Boyer, Erik Bruchez, Leigh Klotz, Steven Pemberton, and Nick Van den Bleeken. W3C. 2014. https://www.w3.org/MarkUp/Forms/wiki/XForms_2.0

[6] *XForms 1.1 Quick Reference*. Steven Pemberton. W3C. 2010. http://www.w3.org/MarkUp/Forms/2010/xforms11-qr.html

[7] *XForms for HTML Authors*. Steven Pemberton. W3C. 2010. http://www.w3.org/MarkUp/Forms/2010/xforms11-for-html-authors

[8] http://www.cwi.nl/~steven/Talks/2014/xml-london

[9] *XSLTForms*. http://www.agencexml.com/xsltforms

A. Credit