A News Carousel in XForms

June 21, 2019

Steven Pemberton (/authors/steven-pemberton/)

Steven Pemberton continues his series on XForms with an example of building a news carousel.

Introduction

At my work, at various locations, there are screens hanging on the wall displaying current news and announcements. They display one item at a time, displaying it for a while, before going on to the next. Each item typically consists of a headline, an image, and some text.

I asked how they implemented it, and they said that the server generates a webpage with a single news item and a timeout in a header. The client screen displays the page, and when it times out, reloads it, which by then has been replaced by a page with the next news item.

Let's do something to the same effect in XForms.

The News

We'll store the news items in a file as a series of `item` elements:

```
<news>
   <item>
      <title>CWI in Business 2019 – A Compass for Digital Innovation</title>
      <image>cwi-in-bedrijf.jpg</image>
      <p>On Thursday 16 May CWI organized its matchmaking and networking
      event CWI in Business (CWI in Bedrijf 2019). Speakers from the
      business community and CWI researchers showed future opportunities
      in digital innovation. Keynote speakers were Jeroen Maas
      (Amsterdam Economic Board) and John Baekelmans (imec/Holst
      Centre).</p>
   </item>
   <item>
      ...
```

We store that data in an instance:

```
<model id="m">
   <instance id="news" src="news.xml"/>
</model>
```

To display a news item, we are going to use a group to select one item at a time:

```
<group ref="instance('news')/item[position()=instance('index')/i]">
```

and at intervals update the value of the index. So we need to add an instance to store the index:

```
<instance id="index">
   <index xmlns="">
      <i>1</i>
      <n/>
   </index>
</instance>
<bind ref="n" calculate="count(instance('news')/item)"/>
```

and we've calculated the total number of news items we have as well.

Now we can start displaying.

```
<group ref="instance('news')/item[position()=instance('index')/i]">
   <label>News</label>
   <output class="image" ref="image" mediatype="image/*"/>
   <output class="title" ref="title"/>
   <repeat ref="p">
      <output class="p" ref="."/>
   </repeat>
</group>
```

This displays the image, the title, and the paragraphs of the text. A style sheet does the styling.

Because of how bindings to controls work in XForms, if a news item has no image, none will be displayed.

The only other thing we need to do is update the index at regular intervals. At startup we dispatch an event that we shall call `tick`:

```
<action ev:event="xforms-ready">
   <dispatch targetid="m" name="tick" delay="10000"/>
</action>
```

The delay is in milliseconds, so this sends the event after 10 seconds (too short in real life, but OK for the purposes of this example).

When the event arrives after the ten seconds, we catch it, update the index, and dispatch a new event:

```
<action ev:event="tick">
   <setvalue ref="instance('index')/i" value="(. mod ../n) + 1"/>
   <dispatch targetid="m" name="tick" delay="10000"/>
</action>
```

The expression `(. mod ../n) + 1` increments the index up to and including the number of items there are, and then resets it to one.

The result, with suitable CSS, looks like this (I've added a count of the items as well):

News
3 of 3
Steven Pemberton in Nieuwsuur TV On Web@30

To celebrate 30 years since Tim Berners-Lee's proposal for the World Wide Web, an anniversary event is webcast on 12 March. In the Netherlands, CWI Web pioneer Steven Pemberton was interviewed by Nieuwsuur. It was broadcast on national TV on 8 March.

Source (../forms/examples/xmlcom/news/news1.xhtml)

Mixed Content

This only accepts plain text for the paragraphs. Even if the paragraphs contain other elements, they will have no effect on the output. For instance, because of how `output` works in XForms, this:

```
<p>This <b>will</b> only show up as plain text</p>
```

will look identical to this:

```
<p>This will only show up as plain text</p>
```

To fix that, we repeat over the *nodes* under the `p` elements, and output them differently if they are an element we recognise:

```
<repeat ref="p">
   <repeat ref="node()">
      ... output the node in some sort of way ...
   </repeat>
</repeat>
```

For instance, we can output text nodes, plus `i` and `b` elements by looking at the node's name:

```
<repeat ref="p">
   <repeat ref="node()">
      <output class="text" ref=".[name(.)='#text']"/>
      <output class="i" ref=".[name(.)='i']"/>
      <output class="b" ref=".[name(.)='b']"/>
   </repeat>
</repeat>
```

The idiom `ref=".[name(.)='b']"` only selects the node if its name is `b`. If it isn't, nothing gets output by this. Text nodes have a special name `#text`.

Here's what it looks like:

News
3 of 3
Steven Pemberton in Nieuwsuur TV On Web@30

To celebrate 30 years since Tim Berners-Lee's proposal for the World Wide Web, an anniversary event is webcast on 12 March. In the Netherlands, CWI Web pioneer**Steven Pemberton**was interviewed by*Nieuwsuur*. It was broadcast on national TV on 8 March.

Source (../forms/examples/xmlcom/news/news2.xhtml)

Other ways

If you then want to add another element you can just add another `output` along with a stylesheet rule to match:

```
<output class="code" ref=".[name(.)='code']"/>
```

However, there is a way to put them all together in a single output:

```
<output class="{name(.)}" ref=".[name(.)='i' or name(.)='b' or name(.)='code']"/>
```

and in fact we can simplify this even further, by allowing all nodetypes, as long as we do something special for the nodes starting with #. If we change the expression for `class` from

```
class="{name(.)}"
```

to

```
class="{if(substring(name(.), 1, 1)='#',
           substring(name(.), 2),
           name(.))}"
```

then text nodes will be displayed with the style rule `text`, and so on, and we can write:

```
<repeat ref="p">
   <repeat ref="node()">
      <output class="{if(substring(name(.), 1, 1)='#',
                         substring(name(.), 2),
                         name(.))}"
              ref="."/>
   </repeat>
</repeat>
```

Then to add support for a new element, you only have to add a style rule for it.

Renewing

One other thing that needs to be done is to periodically refresh the news instance, otherwise we'd be displaying the same news for ever.

The way to do it is to use a *submission* that replaces the instance:

```
<submission id="renew" resource="news.xml"
            method="get" serialization="none"
            replace="instance" instance="news"/>
```

When activated this will get the news file again, and replace the contents of the news instance with it: `resource` is the file we want to get, `method` is the http protocol we want to use (`get` is the default so we didn't actually have to mention it), `serialization="none"` means that we are sending no data to the server, `replace="instance"` says that the returned data should replace an instance, and `instance` says which one.

We only have to decide when to activate it. The simplest method is to use another timer, and just renew at intervals. Add another event at startup with a delay of ten minutes:

```
<action ev:event="xforms-ready">
   <dispatch targetid="m" name="tick"  delay="10000"/>
   <dispatch targetid="m" name="renew" delay="600000"/>
</action>
```

and catch it when it goes off, activate the submission, and dispatch the next timer event:

```
<action ev:event="renew">
   <send submission="renew"/>
   <dispatch targetid="m" name="renew" delay="600000"/>
</action>
```

However, it would be tidier if the renewal happened synchronously with the display of news items. To do this, we still keep the timer we just created, but when it goes off, rather than doing the submission, we just record that a submission needs to be done:

```
<action ev:event="renew">
   <setvalue ref="instance('index')/renew">yes</setvalue>
   <dispatch targetid="m" name="renew" delay="600000"/>
</action>
```

This clearly needs a new value in the `index` instance:

```
<instance id="index">
   <index xmlns="">
      <i>1</i>
      <n/>
      <renew>no</renew>
   </index>
</instance>
```

Then the display loop can check at the end of each loop if the news instance needs renewing, when we catch a `tick` event:

```
<action ev:event="tick">
   <setvalue ref="instance('index')/i" value="(. mod ../n) + 1"/>
   <send submission="renew" if="instance('index')/i = 1 and instance('index')/renew = 'yes'"
   <dispatch targetid="m" name="tick" delay="10000"/>
</action>
```

The renew value should be reset to `no` , but only once we are sure the submission has successfully finished:

```
<action ev:event="xforms-submit-done">
   <setvalue ref="instance('index')/renew">no</setvalue>
</action>
```

You won't see much difference in this final version, since the news file isn't changing, but you can see the value of `renew` get updated every 60 seconds (actually I set it to 61 seconds: that way it gets set just after displaying the first news item for the third time, and reset when displaying it for the fourth time). The news file really does get re-read.

News
3 of 3 renew: yes
Steven Pemberton in Nieuwsuur TV On Web@30

To celebrate 30 years since Tim Berners-Lee's proposal for the World Wide Web, an anniversary event is webcast on 12 March. In the Netherlands, CWI Web pioneer**Steven Pemberton**was interviewed by*Nieuwsuur*. It was broadcast on national TV on 8 March.

Source (../forms/examples/xmlcom/news/news4.xhtml)

Related links

- An Introduction to XForms (/articles/2018/11/27/introduction-xforms/)
- Viewing Data with XForms (/articles/2019/02/08/viewing-data-xforms/)
- A Calendar in XForms (/articles/2019/03/07/calendar-xforms/)
- A Clock in XForms (/articles/2019/04/07/clock-xforms/)
- A Game in XForms (/articles/2019/05/20/game-xforms/)