XForms: Presentation

Contents

Introduction

CSS has a special *presentation* mode:

```
@media projection {
 ...
}
```

When the browser is put into presentation mode, those styling rules apply.

The idea is that the browser goes into full-screen mode, and the projection rules will typically increase the font size, and express where 'page' breaks are.

Advantages

This has had several advantages, for instance:

- it makes content easily repurposable,
- it gives a choice of editing software,
- it gives quite a lot of control,
- it guarantees a long life for the content.

Packages

Unfortunately, no browser supports CSS presentation mode any more.

Packages of Javascript have emerged to support presentation, such as reveal, remark, webslides, deck, and shwr, (and *dozens* more) and although some are very cute, they all have some underlying problems:

- Largely not standardised: each has its own format for slides, and its own package of javascript, so you can't swap between packages.
- To repurpose existing content, you have to edit the files.
- If support for the package disappears, you are in trouble: this is exactly the same problem as with proprietary software.

A presentation application in XForms

Not quite as good as having a standard built into the browser, but the advantages are:

- Very easy; a surprisingly small amount of code;
- You can still repurpose and use existing content;
- All the power of XHTML+CSS for styling.

The Basis

The slide deck is an XHTML document; each slide is a `div` with rather simple XHTML:

```
<div>
   <h2>The Basis</h2>
   <p>The slide deck is an XHTML document;
      each slide is a <code>div</code> with
      rather simple XHTML: </p>
   <pre>... etc ...</pre>
</div>
```

Slides

An initial slide deck is loaded into an instance:

```
<instance src="http://www.cwi.nl/~steven/Talks/...whatever..."/>
```

(we'll see later how to load different decks).

Selecting slides

The central part is a `group` that selects one `div`:

```
<group ref="h:body/h:div[position()=instance('i')/index]">
   ...
</group>
```

This requires an admin instance that will keep track of which slide we are looking at any time, initalised to 1:

```
<instance id="i">
    <admin xmlns="">
       <index>1</index>
    </admin>
</instance>
```

Next slide

We *could* add buttons to step through the slides like this:

```
<trigger>
    <label>←</label>
    <setvalue ev:event="DOMActivate" ref="instance('i')/index" value=". - 1"
</trigger>
<trigger>
    <label>→</label>
    <setvalue ev:event="DOMActivate" ref="instance('i')/index" value=". + 1"
</trigger>
```

Remote

Better to do it via the keyboard, not least because presentation remotes act as if they are keyboards, sending the characters `Page Up` and `Page Down` when the buttons are pressed:

```
<action ev:event="keydown" ev:defaultAction="cancel">
    <setvalue ref="instance('i')/index" value=". - 1"
              if="event('key')='PageUp' or event('key')='ArrowLeft'"/>
    <setvalue ref="instance('i')/index" value=". + 1"
              if="event('key')='PageDown' or event('key')='ArrowRight'"/>
</action>
```

It is necessary to cancel the default action of the event, since otherwise the browser would do a page up or down as well.

Displaying One Slide

Each slide contains a sequence of XHTML elements. So within the `group` holding the slide, we just have to display those elements. We treat each element within the `div`:

```
<repeat ref="*">
   ...
</repeat>
```

and deal with each element separately.

Elements

Here are some simple cases:

```
<output class="h1" ref=".[name(.)='h1']"/>
<output class="h2" ref=".[name(.)='h2']"/>
<output class="pre" ref=".[name(.)='pre']"/>
```

The XPath idiom "`.[name(.)='h1']`" selects the current item only if its name is 'h1'. If its name doesn't match, then no node is selected by the output element, and so it is *disabled* and does nothing; if the name matches, then its content is output.

By attaching a class, CSS controls how it will be displayed. Clearly at most one of the output elements will be enabled.

In fact we can combine these by taking advantage of attribute value templates in XForms 2:

```
<output class="{name(.)}" ref=".[name(.)='h1' or name(.)='h2' or name(.)='pr
```

Compound elements

More complicated cases are those elements that themselves contain other elements, such as `<p>` and `<ul>`.

The easier of these two is `<ul>`. Here we do a similar trick, and repeat over the contained elements, with the advantage that we know they are all `<li>` elements:

```
<group class="ul" ref=".[name(.)='ul']">
    <repeat ref="h:li">
        <output class="li" ref="."/>
    </repeat>
</group>
```

Mixed content

The `<p>` elements have a complication that they may contain mixed content. No worry though, because there is a selector for that. Rather than using `ref="*"` as we did in the outermost repeat, we use `ref="node()"` which repeats over all nodes, which includes text and comments as well as elements:

```
<group class="p" ref=".[name(.)='p']">
    <repeat ref="node()">
        <output class="text" ref=".[name(.)='#text']"/>
        <output class="{name(.)}" ref=".[name(.)='em' or name(.)='strong' or
        <output class="img" ref=".[name(.)='img']" value="concat(instance('i
    </repeat>
</group>
```

Images

Since there is no output element that selects comment nodes, they won't be displayed.

The only interesting case here is for images. The `src` attribute is relative to the original slides, so we have to concatenate it with the base URL of the slides, which we store in the admin instance:

```
<instance id="i">
    <admin xmlns="">
        <base>https://www.cwi.nl/~steven/Talks/..whatever.../</base>
        <index>1</index>
    </admin>
</instance>
```

Result

Here it is in action. You have to click on one of the arrows in the top right to give focus to it; after that you can use the arrow keys on your keyboard:

Minesweeper in XForms

← → 1/18

Steven Pemberton, CWI, Amsterdam

Source

Loading Other Slide Sets

Having the base stored in the admin instance makes it easy. All we need to do is ask the user for the URL of the new slide set, submit it, and replace the slides instance with the result:

```
<input ref="instance('i')/base" label="URL:"/>
<submit submission="change" label="Go"/>
```

where the `<submission>` element looks like this:

```
<submission id="change" resource="{instance('i')/base}" serialize="none" rep
```

However, we need to do one other thing: when the new slides are loaded, we need to reset the index back to 1:

```
<submission resource="{instance('i')/base}" id="change" method="get" seriali
    <action ev:event="xforms-submit-done">
        <setvalue ref="instance('i')/index" value="1"/>
        <toggle case="show"/>
    </action>
</submission>
```