On the Design of a Self-Referential Tutorial

Steven Pemberton, CWI, Amsterdam

XForms [XF1][XF2] is a declarative programming language, with a different programming paradigm compared to traditional programming languages. Since it requires a new way of thinking about programming, learning materials are paramount for people wanting to use the language and understand its benefits.

This paper discusses the method used to build a hands-on tutorial for XForms, using XForms, with the decisions taken, the techniques used, and the benefits that the approach gave.

1. Tutorials

A tutorial is a challenging educational setting, both for those learning and those instructing. Time is severely limited, and there is inevitably far more material on the subject than can possibly be taught in the time available. Unavoidably, compromises have to be made: do you make it deep and narrow, where you learn a subset of the material in great detail, or broad and shallow, where you get an overview of the whole, with less detail? Should it be a teaser to tempt attendees to later self-study, or a starter, so that attendees have at least a working knowledge of some of the material?

And then there is the method of presentation: should it be lecture style, permitting coverage of more of the material, or hands-on, allowing attendees more direct acquaintance with the material, while reducing the coverage?

1.1. XForms Tutorials

Most XForms tutorials [e.g. t1, t2] had to-date been of the lecture style, covering most of the language. However, a request for a hands-on tutorial motivated a new approach.

After attending several hands-on tutorials on other topics, the author concluded that in order to optimally use the time available for exercises, attendees shouldn't be required to start from scratch, since precious time is lost dealing with trivial issues like set up and syntax. Rather, the exercises should all require the attendee to make a change to an existing, working, example, using the newly-acquired knowledge. In that way, you get the advantage of the hands-on approach, while minimising trivial administrative details, with the added advantage of attendees being confronted with larger working examples right from the start.

As a consequence, the tutorial was designed as a rapid-fire sequence of exercises, each consisting of 5 minutes of presentation, followed by 5 minutes of coding. This has the added advantage of maximising the attendees' concentration, thanks to the recurrent switching of activities. The exercises themselves are mostly not stand-alone, but cumulative, each one building on an earlier one, so that at the end the attendee has a handful of small, but in themselves useful, applications. Although the tutorial was designed to be part of a live event, it also supports the use of self-study.

The resulting tutorial [t3] is interesting in that it is not only *about* XForms, but is also built *in* XForms, which in itself gave surprising possibilities.

2. Content

The first task in designing a tutorial is to identify the topics to be taught. This was a fairly easy job in this case, since it only involved running through the headings of the XForms specification, and selecting topics. Naturally enough, a structured XML document listing those topics, and their relative structure was created as part of this process:

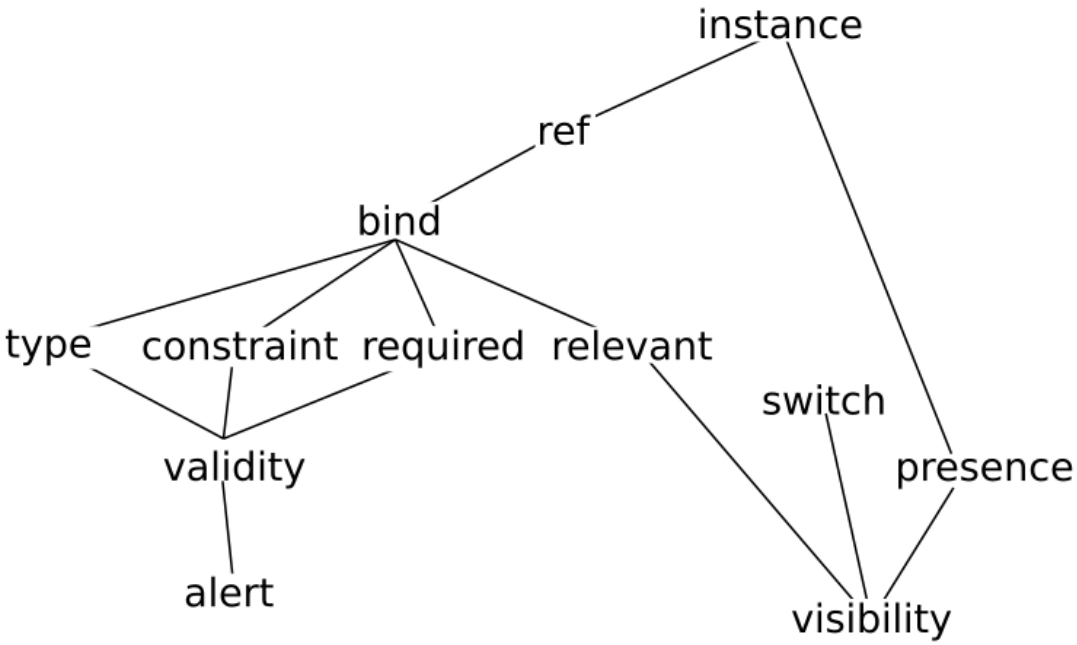
```
<learn title="What you will learn">
   <topic name="Structure of an XForm">
      <topic done="" name="&lt;model>"/>
      <topic done="" name="content"/>
   </topic>
   <topic name="Instance data">
      <topic done="" name="internal &lt;instance>"/>
      <topic done="" name="external &lt;instance @resource>"/>
      <topic done="" name="&lt;bind>">
     <topic done="" name="@ref"/>
     <topic done="" name="@type"/>
     <topic done="" name="@calculate"/>
     <topic done="" name="@relevant"/>
     <topic done="" name="@required"/>
     <topic done="" name="@readonly"/>
     <topic done="" name="@constraint"/>
      </topic>
   </topic>
   ...
```

In fact this one document ended up being used in three ways: firstly and primarily as the list of topics that needed to be covered, that could then be checked off one by one as the tutorial was written, as a check for completeness. But additionally it ended up displayed in two different ways in the tutorial itself: once at the beginning as an overview of what would be taught in the course, and again at the end of the course, though displayed in a different way, as a summary of what had been taught, allowing the reader to check each topic off as a reminder of what had been learned (a correspondent had indicated that tutorials that included such a check-list on average were evaluated by attendees 10% higher). In fact this last section, allowing the attendee to check off topics learned, was the first written, so that as the tutorial was developed, the topics could be checked off.

The next step is to decide the *order* these topics need to be taught. This was done by creating a dependency graph: to understand this topic, you need to know about these topics, to understand *those* topics, you need to know about these other topics, and so on. This gives a partial ordering of topics, which can then be ordered linearly at will.

For example, to understand the `alert` element, you need to know about validity; to understand validity, you need to know about `type`, `constraint`, and `required`, to know about those, you need to know about `bind`, and `ref`, to know about those you need to know about `instance`.

Figure 1. A simplified partial dependency graph



It is worth pointing out that there is some leeway in even the partial ordering that the dependency graph generates, because rather than completely covering all the antecedents of a topic first, you can use progressive disclosure [pd]; for instance you could partially cover validity, talking only about types, and then extend it later with constraints and `required`. What the graph does reveal though is the order the partial disclosure has to happen.

This then leads to a fundamental decision: should the topics be taught top-down, or bottom-up? Bottom-up starts at the leaves of the dependency graph and works upwards building higher abstractions out of the lower-level ones; top-down works in the other direction, starting with the high-level abstractions, and working down towards the leaves. I am not aware of studies of the relative effectiveness of the two methods, nor whether one is more effective than the other with certain types of audience. I personally believe, based on experience of teaching, that top-down works better in general, especially with knowledgeable audiences, since it gives more opportunity to see the wood for the trees: each new thing you learn, you understand how it fits in the larger picture.

3. Structure

The topics were largely treated one per chapter, and in general written in the order they would be presented. It was attempted to make each chapter not more than one screen-full long, around 50-60 lines of text, followed by its exercise.

Each chapter was built as a single XHTML+XForms document; if it became too long, it could be split into two chapters; occasionally, if a chapter was very short, or if context demanded it, two concepts would be covered in a single chapter.

An XML document was kept as chapters were written documenting the title, filename, concepts covered, name of the exercise file, and name of the example answer file:

```
<tutorial>
   <entry>
      <title>Introduction</title>
```

```
      <file>intro.xhtml</file>
      <concept>implementations</concept>
      <concept>W3C standard</concept>
      <concept>future</concept>
   </entry>
   <entry>
      <title>What you will learn</title>
      <file>learn.xhtml</file>
      <concept>quick reference</concept>
   </entry>
   <entry>
      <title>The structure of an XForm</title>
      <file>structure.xhtml</file>
      <concept>structure</concept>
      <concept>model</concept>
      <concept>instance</concept>
      <concept>controls</concept>
      <concept>XML</concept>
      <concept>media type</concept>
   </entry>
   <entry>
      <title>Data Instances and the Input Control</title>
      <file>input.xhtml</file>
      <exercise>input-exercise.xhtml</exercise>
      <answer>input-answer.xhtml</answer>
      <concept>&lt;instance resource=""/></concept>
      <concept>&lt;bind ref=""/></concept>
      <concept>@type</concept>
      <concept>integer type</concept>
      <concept>date type</concept>
      <concept>string type</concept>
      <concept>boolean type</concept>
      <concept>&lt;input ref=""/> control</concept>
      <concept>&lt;label></concept>
      <concept>@incremental</concept>
   </entry>
   <entry>
      <title>The output control</title>
      <file>output.xhtml</file>
      <exercise>output-exercise.xhtml</exercise>
      <answer>output-answer.xhtml</answer>
      <concept>&lt;output ref=""/> control</concept>
      <concept>@value</concept>
      <concept>XPath / @ *</concept>
      <concept>count() function</concept>
   </entry>
   ...
```

4. Navigation

This document of chapters was then used as the basis of the navigation part of the tutorial. This was modelled on the navigation used in an earlier XForms application, the XForms 2.0 test suite [ts].

A navigation instance was created containing

1. a value `base` that defines where the chapter files are stored,
2. a value for the filename of the chapter that is currently being read,
3. a value that constructs the required URL from the base and the filename:

```
<instance id="nav">
   <nav xmlns="">
      <base>chapters/</base>
      <file/>
      <url/>
   </nav>
</instance>
<bind ref="instance('nav')/url" calculate="concat(../base, ../file)"/>
```

To set the filename, a `select1` control is populated using the tutorial chapters instance: the labels are the title of the chapter, and the value is the file name of that chapter:

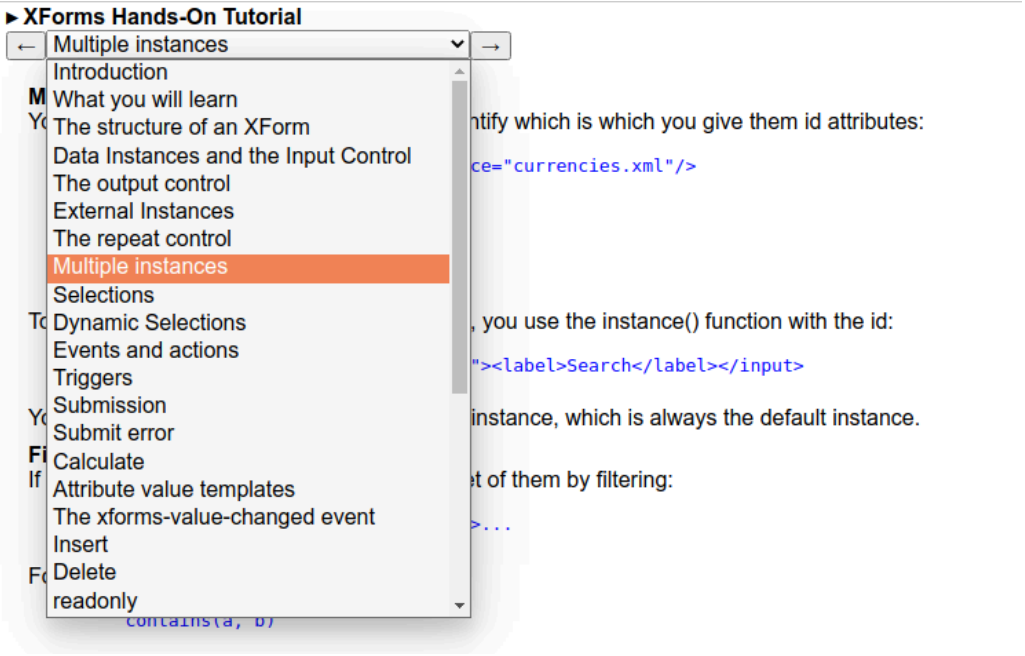
```
<select1 ref="instance('nav')/file">
   <itemset ref="instance('tut')/entry">
       <label ref="title"/>
       <value value="file"/>
   </itemset>
</select1>
```

Either side of this control, are triggers that decrement and increment this value (with judicious use of `preceding-sibling` and `following-sibling`):

```
<trigger
      ref="instance('tut')/entry[file=instance('nav')/file]/preceding-sibling::entry[1]">
   <label>←</label>
   <hint><output ref="title"/></hint>
   <setvalue ref="instance('nav')/file"
             value="context()/file"
             ev:event="DOMActivate"/>
</trigger>
```

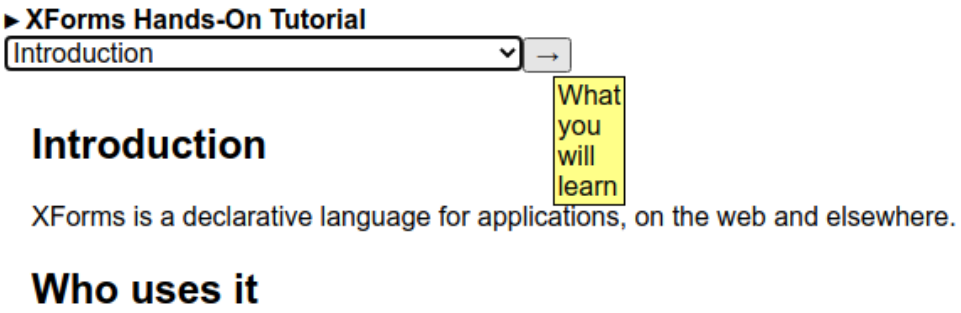
In this way, the user can select a chapter either from the drop down, or by navigating from chapter to chapter using the arrows.

Selecting a chapter



Note that thanks to how the `ref` on the trigger works in XForms, in this case referencing the previous chapter, if there is no previous chapter, the trigger will not be displayed.

The initial chapter, with no previous



To display the content of the chapter, an html iframe is used:

```
<html:iframe id="chapter" src="{instance('nav')/url}"></html:iframe>
```

Thanks to how XForms works, whenever the `file` value gets changed, the `url` value automatically gets updated, and so the new chapter is displayed.

The upshot of all this is that the tutorial instance doesn't *describe* the structure of the tutorial but actually *defines* it. If a chapter needs to be moved, you just move it in the tutorial instance; if a new chapter is written, add it to the tutorial document. The navigation will consequently automatically be updated, and the chapter can be displayed.

5. Chapters

Each chapter is thus a separate XForms document. This has amazing advantages when talking about XForms, because the text can actually contain the examples it is describing, rather than descriptions of the expected output.

For example, if a chapter contains the text:

By default, outputs directly after each other will abut. If a is 3, and b is 4, then

```
<output ref="a"/><output ref="b"/>
```

will output:

```
34
```

the underlying code is:

```
By default, outputs directly after each other will abut.
If a is <output  ref="a"/>, and b is <output ref="b"/>, then
<group class="pre">
   &lt;output ref="a"/>&lt;output ref="b"/>
</group>
will output:
<group class="pre">
   <output ref="a"/><output ref="b"/>
</group>
```

In other words a really *is* 3, and b really *is* 4, and the example output is produced by the code itself.

This of course has immense advantage: you make fewer mistakes, you can put decisions off, or make changes easily without worrying about consistency between text and results.

6. Exercises

The exercises are based on 5 example applications: a short initial ‘toy’ one covering two exercises introducing the basic concepts such as instances, types, input and output, and then two large ones each covering 10 exercises, both broken in the middle by two small examples. The large examples helped with ordering the partial order of the material, since concepts could be introduced as they were needed for enlarging the functionality of the applications.

The first large example introduces external instances loaded over the internet, in this case the exchange values of a large number of currencies, and builds an application to display the exchange rate between two currencies of choice.

It is broken in the middle with a small application to show how times are manipulated, and to introduce the concept of events.

Submission is then introduced to show how to to load new instances, and how to deal with submission errors, and then how to generate a URL for use in submission.

The second application is a to-do list, in order to show how to manipulate lists, how to use submission to save data, how to detect when data needs saving, and finally how to automatically save data when it has changed.

It is broken in the middle with a small log-in application, that shows how to input passwords, and how to deal with validity errors.

In some senses, finding suitable examples that were expandable in this way, gradually adding new concepts as they were needed was the hardest part of designing the tutorial. It is easy to create one tiny toy example for each concept being taught, but more work to

design ones that are meaningful and useful applications that have the right collection of necessary concepts.

7. Server

One of the requirements for following the tutorial is that the user have access to an HTTP server, and furthermore, one that supports the PUT method [put], since several of the exercises use it. For reasons that are not entirely clear, most HTTP servers available don't support PUT at all, or otherwise only with a lot of extra work. So the options for the tutorial apparently are: require the attendees to locate and install an existing server that accepts PUT; run a server and then require anybody who wants to use the tutorial to acquire a username and login for it; or supply a minimal server that the user can easily install and use. We opted for the third.

Luckily, writing a simple server without the normal industrial-strength requirements of large-scale servers is fairly easy, and since we had already written one for the XForms Test Suite [ts], XContents [xc], we used that, a few hundred lines of C code, which we then compiled for a number of standard platforms. Unfortunately, during testing, we discovered that on Windows, one virus scanner falsely identified the server as malware, and so for the people who ran against that problem we also wrote a JavaScript version which runs under Node.

Since the tutorial demands a server that uses PUT, the tutorial itself could use it for its own purposes. In particular whenever the user navigates to a new chapter, the current state is saved so that should the user stop and log out, on return the tutorial is restarted at the last-used location.

8. Experience

At the time of writing, the tutorial has been given once, not counting a couple of try-outs with single victims.

With only 2 hours allocated, and each exercise being budgeted for 5 minutes presentation and 5 minutes working, only 12 of the 25 sections were handled. Nevertheless, this wasn't seen as a problem, since the tutorial is also designed for self-study, and so the attendees could do the rest of the tutorial at their leisure.

There is a problem with doing a tutorial remotely, as was the case, that there is far less contact with the individual attendees, in order to solve problems, and less opportunity to assess if people have properly finished an exercise, especially since later exercises use the results of previous exercises. In fact this could be seen as a disadvantage of the cumulative approach to the exercises, in that it requires previous exercises to be completed before you can continue.

The tutorial was well-received; it revealed a few places where the exposition could be clarified, but otherwise went well.

9. Conclusion

Tutorial design, like any curriculum design, is hard! It is a surprising amount of work, with a lot of internal dependencies. However I have to admit that writing this one was a lot of fun, being about and in XForms allowed me to submerge myself fully in the subject, and allowed a lot of the internal dependencies to take care of themselves. Using XForms for the navigation introduced a large amount of flexibility in the construction of the tutorial.

Bibliography

[t1] Steven Pemberton. *Declarative Applications with XForms*. XML Prague. 2020. https://homepages.cwi.nl/~steven/Talks/2020/02-13-xforms/ [https://homepages.cwi.nl/%7Esteven/Talks/2020/02-13-xforms/] .

[t2] Steven Pemberton. *Web Applications with XForms 2.0*. WWW 2013. 2013. https://homepages.cwi.nl/~steven/Talks/2013/05-14-webapps-xforms2/ [https://homepages.cwi.nl/%7Esteven/Talks/2013/05-14-webapps-xforms2/] .

[t3] Steven Pemberton. *XForms Hands On*. Declarative Amsterdam. 2020. https://homepages.cwi.nl/~steven/Talks/2020/10-08-tutorial/ [https://homepages.cwi.nl/%7Esteven/Talks/2020/10-08-tutorial/] .

[xf1] John Boyer. *XForms 1.1*. 2009. W3C. https://www.w3.org/TR/xforms11 .

[xf2] Erik Bruchez et al. (eds). *XForms 2.0*. W3C. 2021. https://www.w3.org/community/xformsusers/wiki/XForms_2.0 .

[put] T. Berners-Lee et al.. *Hypertext Transfer Protocol -- HTTP/1.0 (appendix D.1.1)*. . IETF. 1996. https://tools.ietf.org/html/rfc1945#appendix-D.1.1 .

[ts] Steven Pemberton. *The XForms 2.0 Test Suite*. MarkupUK. 2018. https://markupuk.org/2018/webhelp/index.html#ar14.html .

[xc] Steven Pemberton. *XContents Minimal Web Server*. CWI. 2020. https://homepages.cwi.nl/~steven/xcontents/xcontents.c [https://homepages.cwi.nl/%7Esteven/xcontents/xcontents.c] .

[pd] Frank Spillers. *Progressive Disclosure, in Soegaard and Dam (eds.), The Glossary of Human Computer Interaction, Chapter 44*. The Interaction Design Foundation. undated. https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/progressive-disclosure .