

XForms

Improving the Web Forms Experience

A CHI 2004 Tutorial

Steven Pemberton

CWI and W3C
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

Steven.Pemberton@cwi.nl

www.cwi.nl/~steven

Table of Contents

- ii. Agenda
- iii. About the Instructor
- iv. Objectives
 - 1. Block 1
 - 20. Block 2
 - 33. Block 3
 - 47. Block 4

Agenda

The day is split into four blocks, each of 90 minutes.

Each block consists of about 45 minutes lecture, followed by 45 minutes practical.

The breaks between blocks are 30 minutes, with 90 minutes for lunch.

Block 1

Introduction, equivalents for HTML features.

Break

Block 2

Types, bindings and constraints.

Lunch

Block 3

Instance data, XPath and submission.

Break

Block 4

Events, wizards and shopping baskets.

About the Instructor

Steven Pemberton is a researcher at the CWI, The Centre for Mathematics and Computer Science, a nationally-funded research centre in Amsterdam, The Netherlands, the first non-military Internet site in Europe.

Steven's research is in interaction, and how the underlying software architecture can support the user. At the end of the 80's he built a style-sheet based hypertext system called Views.

Steven has been involved with the World Wide Web since the beginning. He organised two workshops at the first World Wide Web Conference in 1994, chaired the first W3C Style Sheets workshop, and the first W3C Internationalisation workshop. He was a member of the CSS Working Group from its start, and is a long-time member (now chair) of the HTML Working Group, and co-chair of the XForms Working Group. He is co-author of (amongst other things) HTML 4, CSS, XHTML and XForms.

Steven is also Editor-in-Chief of ACM/interactions.

Objectives

HTML Forms, introduced in 1993, were the basis of the e-commerce revolution. After 10 years experience, it has become clear how to improve on them, for the end user, the author, and the owners of the services that the forms are addressing. XForms is a new technology, announced in October 2003, intended to replace HTML Forms.

The advantages of XForms include:

- It improves the user experience: XForms has been designed to allow much to be checked by the browser, such as types of fields being filled in, or that one date is later than another. This reduces the need for round trips to the server or for extensive script-based solutions, and improves the user experience by giving immediate feedback to what is being filled in.
- It is XML, and it can submit XML.
- It combines existing XML technologies: Rather than reinventing the wheel, XForms uses a number of existing XML technologies, such as XPath for addressing and calculating values, and XML Schemas for defining data types. This has a dual benefit: ease of learning for people who already know these technologies, and implementors can use off-the-shelf components to build their systems.
- It is internationalized.
- It is accessible: XForms has been designed so that it will work equally well with accessible technologies (for instance for blind users) and with traditional visual browsers.
- It is device independent: The same form can be delivered without change to a traditional browser, a PDA, a mobile phone, a voice browser, and even some more exotic emerging clients such as an Instant Messenger. This greatly eases providing forms to a wide audience, since forms only need to be authored once.
- It is easier to author complicated forms.

The presenter is one of the authors of the XForms specifications, and is chair of the Forms Working Group that produced the technology.

This tutorial works from a basis of HTML Forms, and introduces XForms step-by-step. It covers essentially all of XForms except some technical details about events, and no more than a passing reference to the use of Schemas.

Emphasis is on how to improve the user experience, and how XForms improves accessibility and device independence, and makes the author's life easy in producing a better experience.

Plan

- 9.00-9.45 Introduction. HTML equivalents
- 9.45-10.30 Practical: Converting HTML to XForms
- 10.30-11.00 **Break**
- 11.00-11.45 Types; bind; constraints
- 11.45-12.30 Practical: Adapting to earlier answers
- 12.30-13.30 **Lunch**
- 13.30-14.15 Instance data; XPath; submission variants.
- 14.15-15.00 Practical: Editing XML with XForms
- 15.00-15.30 **Break**
- 15.30-16.15 Events, shopping baskets and wizards
- 16.15-17.00 Practical: A shopping basket-style form.
- 17.00 **End**

Forms

- Forms have been the basis of the e-commerce revolution
- You find them everywhere on the web: search, login, email, shopping, blogs, wiki, configuring hardware, ...
- HTML Forms are really quite simple, and it is quite startling how much has been done with them
- But they have some problems

Problems with HTML Forms

- Presentation oriented
- No types
- Reliance on scripting
- Problems with non-Western scripts
- Ping-ponging to the server
- Impoverished data-model
- Hard to manage, hard to see what is returned
- No support for wizards and shopping carts etc.

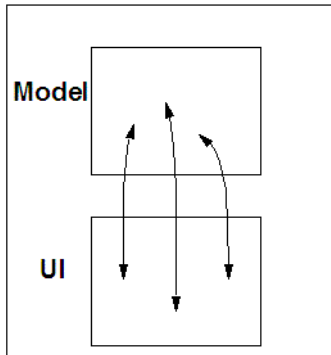
Soundbite: "Javascript accounts for 90% of our headaches in complex forms, and is extremely brittle and unmaintainable."

XForms

- e-forms are a current hot topic.
- The W3C XForms working group started out as a sub-group of the HTML working group, but soon spun off as an independent group when it emerged how much work there was to be done.
- XForms has been designed based on an analysis of HTML Forms, what they can do, and what they can't.

The Approach

- The essence is to separate what is being returned from how the values are filled in.
- The **model** specifies an **instance** (the values being collected) and details about them:
 - Types, restrictions
 - Initial values, Relations between values
- The body of the document then **binds** forms controls to values in the instance



Advantages

- **XForms improves the user experience:**
XForms has been designed to allow much to be checked by the browser, such as types of fields being filled in, or that one date is later than another. This reduces the need for round trips to the server or for extensive script-based solutions, and improves the user experience by giving immediate feedback to what is being filled in.
- **It is XML, and it can submit XML:**
More and more data is in XML; XForms allows you access to that data in a uniform manner.

... Advantages

- **It combines existing XML technologies:**

Rather than reinventing the wheel, XForms uses a number of existing XML technologies, such as XPath for addressing and calculating values, and XML Schemas for defining data types. This has a dual benefit: ease of learning for people who already know these technologies, and implementors can use off-the-shelf components to build their systems.

- **It is internationalized:**

Thanks to using XML, there are no problems with loading and submitting non-Western data.

- **It is accessible:**

XForms has been designed so that it will work equally well with accessible technologies (for instance for blind users) and with traditional visual browsers.

... Advantages

- **It is device independent:**

The same form can be delivered without change to a traditional browser, a PDA, a mobile phone, a voice browser, and even some more exotic emerging clients such as an Instant Messenger. This greatly eases providing forms to a wide audience, since forms only need to be authored once.

- **It is easier to author complicated forms:**

By design it supports things that can only be done with scripting in HTML Forms.

Basic structure of XForms

Take this simple HTML form:

```
<html>
<head><title>Search</title></head>
<body>
  <form action="http://example.com/search"
        method="get">
    Find <input type="text" name="q">
    <input type="submit" value="Go">
  </form>
</body>
</html>
```

The main difference in XForms is that details of the values collected and how to submit them are gathered in the head, in an element called `model`; only the form controls are put in the body.

... basic structure

So in this case the minimum you need in the head is (XForms elements and attributes are in lower case):

```
<model>
  <submission
    action="http://example.com/search"
    method="get"
    id="s"/>
</model>
```

The `<form>` element is now no longer needed; the controls in the body look like this:

```
<input ref="q"><label>Find</label></input>
<submit submission="s">
  <label>Go</label>
</submit>
```

... basic structure

What you can hopefully work out from this is that form controls have a `<label>` element as child, the `<input>` uses "ref" instead of "name", and there is a separate `submit` control that links to the details of the submission in the head. So the complete example is:

Complete XForms search example

```
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns="http://www.w3.org/2002/xforms">
<h:head>
  <h:title>Search</h:title>
  <model>
    <submission
      action="http://example.com/search"
      method="get" id="s"/>
    </model>
</h:head>
<h:body>
  <h:p>
    <input ref="q"><label>Find</label></input>
    <submit submission="s"><label>Go</label>
  </submit>
  </h:p>
</h:body></h:html>
```

Namespaces

- Another obvious difference is the use of h: prefixes
- Nothing to do with XForms, but with XML
- Namespace says which language elements belong to
- One language may be the 'default' language
- In the above XForms is the default
- XHTML could have been made the default:

HTML as default namespace

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://www.w3.org/2002/xforms">
<head><title>Search</title>
  <f:model>
    <f:submission method="get" id="s"
      action="http://example.com/search"/>
  </f:model>
</head>
<body>
  <p><f:input ref="q">
    <f:label>Find</f:label>
    </f:input>    <f:submit submission="s">
    <f:label>Go</f:label>
    </f:submit>
  </p>
</body></html>
```

Choice of prefixes

- You can make nothing the default, and prefix all elements
- You can choose any prefix; h: or x: or html: or form:, or ...

Making XForms documents interoperable

- Unfortunately, Microsoft Internet Explorer does not allow you to choose default language
- HTML must be default
- Some magic words are needed, but your documents will also run on true XML systems

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://www.w3.org/2002/xforms">
<head>
  <object width="0" height="0" id="FormsPlayer"
          classid="CLSID:4D0ABA11-C5F0-4478-991A-375C4B648F58">
    <strong>FormsPlayer failed to load</strong>
  </object>
  <?import namespace="f"
            implementation="#FormsPlayer"?>
```

XForms equivalents for simple HTML Forms features

Now to compare one for one HTML forms controls with XForms equivalents

Simple Text Input

To input a single text element

```
First name: <input type="text" name="fname">
```

is written

```
<input ref="fname"><label>First name:</label>
</input>
```

There is no need to indicate that it is text: in the absence of any other information, by default it is text (called *string* in XForms).

First name:

We will see later how to give any control an initial value.

Textarea

To input multiline text

```
Message:  
<textarea name="message" rows="20" cols="80">  
</textarea>
```

is written

```
<textarea ref="message"><label>Message:</label>  
</textarea>
```

Message:

Styling controls

Styling is done using a style sheet. For instance:

```
textarea[ref="message"]  
{ font-family: sans-serif;  
  height: 20em; width: 80em }
```

or

```
textarea[ref="message"]  
{ font-family: serif;  
  height: 2cm; width: 20% }
```

If you want all your textareas to have the same dimensions, you can use

```
textarea { font-family: sans-serif;  
           height: 20em; width: 80em }
```

Adding a stylesheet

The easiest way to include a style sheet in your document is to add this at the beginning of the document:

```
<?xml version="1.0"?>  
<?xml-stylesheet href="style.css"  
                type="text/css"?>
```

where 'style.css' is the name of your stylesheet, although in XHTML you can also say in the <head>:

```
<link rel="stylesheet" type="text/css"  
      href="style.css"/>
```

Radio Buttons

Radio buttons select one value from a set:

```
Gender:  
<input type="radio" name="sex" value="M"> Male  
<input type="radio" name="sex" value="F"> Female
```

becomes

```
<select1 ref="sex">  
  <label>Gender:</label>  
  <item>  
    <label>Male</label><value>M</value>  
  </item>  
  <item>  
    <label>Female</label><value>F</value>  
  </item>  
</select1>
```


Presentation of controls

- Controls principally represent their *purpose*
- `select` and `select1` may be presented as radio buttons, a (scrollable) select area, or a menu.
- Use the hint `appearance="full"` to suggest presentation as radio buttons.
- Use `appearance="compact"` to suggest a select area
- Use `appearance="minimal"` to suggest a menu

Gender: Male Female

Gender:

We will see later how to preselect a value.

Checkboxes

HTML Checkboxes select zero or more from a list.

```
Flavors:  
<input type="checkbox" name="flavors" value="v"> Vanilla  
<input type="checkbox" name="flavors" value="s"> Strawberry  
<input type="checkbox" name="flavors" value="c"> Chocolate
```

is written

.. checkboxes

```
<select ref="flavors" appearance="full">
  <label>Flavors:</label>
  <item>
    <label>Vanilla</label><value>v</value>
  </item>
  <item>
    <label>Strawberry</label><value>s</value>
  </item>
  <item>
    <label>Chocolate</label><value>c</value>
  </item>
</select>
```

Flavors: Vanilla Strawberry Chocolate

Menus

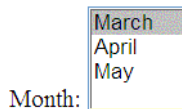
Depending on the presence of the `multiple` attribute in HTML, menus select one, or zero or more from a list of options. You either use `<select1>` to select a single choice, or `<select>` to select zero or more.

```
Month:
<select multiple name="spring">
  <option value="Mar">March</option>
  <option value="Apr">April</option>
  <option>May</option>
</select>
```

would be written:

... menus

```
<select ref="spring" appearance="compact">
<label>Month:</label>
<item>
  <label>March</label><value>Mar</value>
</item>
<item>
  <label>April</label><value>Apr</value>
</item>
<item>
  <label>May</label><value>May</value>
</item>
</select>
```



If multiple isn't on the HTML select, use select1 instead.

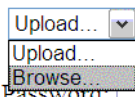
File Select

To select a file to be uploaded

```
<form method="post"
  enctype="multipart/form-data" ...>
...
File: <input type="file" name="attachment">
```

is written

```
<submission method="form-data-post" .../>
...
<upload ref="attachment">
  <label>File:</label>
</upload>
```



Passwords

```
Password: <input type="password" name="pw">
```

is written

```
<secret ref="pw">  
  <label>Password:</label>  
</secret>
```

Password:

Reset

- Hardly anyone actually uses reset buttons
- Yet very many Web forms include them.
- Often the reset button with the text "Reset" is larger than the submission button that is often marked "OK" (and there's no undo)
- While it is possible to create a reset button in XForms, it is deliberately harder to do:

```
<input type="reset">
```

is therefore written

```
<trigger>  
  <label>Clear all fields</label>  
  <reset ev:event="DOMActivate"/>  
</trigger>
```

Clear all fields

Buttons

Buttons have no predefined behavior, but have a behavior attached to them which is triggered when a relevant event occurs.

The button element

```
<input type="button" value="Show"
      onclick="show()">
```

can be written

```
<trigger><label>Show</label>
  <h:script ev:event="DOMActivate"
            type="text/javascript">show()
  </h:script>
</trigger>
```

or

... buttons

```
<trigger ev:event="DOMActivate"
        ev:handler="#show">
  <label>Show</label>
</trigger>
```

where "#show" locates the element (for instance a `script` element) that implements the behavior:

```
<script id="show" ...>...
```

XForms has a number of built in actions that can be executed by a button; see the reset button above for an example.

... buttons

The fact that the `event` attribute has a prefix, means that you have to add the following XML Namespace to the head:

```
xmlns:ev="http://www.w3.org/2001/xml-events"
```

We will be dealing more with events later.

Image Buttons

```
<input type="image" src="..." ...>
```

is written by putting an image into the `<label>` element:

```
<trigger...><label><h:img src="star.gif" .../>
</label></trigger>
```

or by specifying it in a stylesheet

```
<trigger id="activate" ...>
```

with a stylesheet rule

```
trigger#activate {
  background-image: url(button.png);
  background-repeat: none}
```



(Likewise for `<submit>`.)

Optgroup

```
Drink:
<select name="drink">
  <option selected value="none">None</option>
  <optgroup label="Soft drinks">
    <option value="h2o">Water</option>
    <option value="m">Milk</option>
    <option value="oj">Juice</option>
  </optgroup>
  <optgroup label="Wine and beer">
    <option value="rw">Red Wine</option>
    <option value="ww">White Wine</option>
    <option value="b">Beer</option>
  </optgroup>
</select>
```

is written

... optgroup

```
<select1 ref="drink">
  <label>Drink:</label>
  <item><label>None</label><value>none</value></item>
  <choices>
    <label>Soft drinks</label>
    <item><label>Water</label><value>h2o</value></item>
    <item><label>Milk</label><value>m</value></item>
    <item><label>Juice</label><value>oj</value></item>
  </choices>
  <choices>
    <label>Wine and beer</label>
    <item><label>Red wine</label><value>rw</value></item>
    <item><label>White wine</label><value>ww</value></item>
    <item><label>Beer</label><value>b</value></item>
  </choices>
</select1>
```

Grouping Controls

```
<fieldset>
  <legend>Personal Information</legend>
  Last Name: <input name="lastname" type="text">
  First Name: <input name="firstname" type="text">
  Address: <input name="address" type="text">
</fieldset>
```

is written

```
<group>
  <label>Personal Information</label>
  <input ref="lastname"><label>Last name:</label></input>
  <input ref="firstname"><label>First name:</label></input>
  <input ref="address"><label>Address:</label></input>
</group>
```

Note the consistent use of <label>.

Hidden Controls

As you will see shortly, there is no need for hidden controls in XForms.

Practical: Converting an existing HTML Form

Take the file `practical1.html` and view it in your browser. Now create an equivalent XForm

You will find a template file `template.xfm` that you can copy to start from

Output Controls

XForms has two controls that are not in HTML, `output` and `range`.

The `output` control allows you to include values as text in the document.

```
Your current total is: <output ref="sum"/>
```

or

```
<output ref="sum"><label>Total</label></output>
```

This can be used to allow the user to preview values being submitted.

... output

You can also calculate values:

```
Total volume:  
<output value="height * width * depth"/>
```

(where height, width and depth are values collected by other controls.)

Range Controls

This control allows you to specify a constraint on a value.

```
<range ref="volume"  
  start="1" end="10" step="0.5"/>
```

A user agent may represent this as a slider or similar.

Submitted Values

- The attribute named `ref` on each control actually refers to a child of an `instance` element in the model, where the values are gathered before submission.
- If there is no instance element there (as in the search example above), then one is silently created.

Making the Submitted Values Explicit

It is good practice to include an explicit instance, like this for the search example:

```
<model>
  <instance>
    <data xmlns=""><q/></data>
  </instance>
  <submission
    action="http://example.com/search"
    method="get" id="s"/>
</model>
...
<input ref="q">
  <label>Search</label>
</input>
```

... explicit values

- You immediately see that the only data value submitted is called "q".
- The system will now also check that when you say `ref="q"` that there really is a `q` in the instance.
- It is essential that you put the `xmlns=""` on your instance data, to tell the processor that the elements here are neither XHTML nor XForms elements.
- We've used the tag `<data>` here, but you can choose any tag you like.

Initial Values

For initialising controls including initialising checked boxes, and selected menu items etc., you just supply an instance with pre-filled values. For the search example:

```
<instance>  
  <data xmlns=""><q>Keywords</q></data>  
</instance>
```

would pre-fill the text control with the word *Keywords*.

... initial values for checkboxes

```
<select ref="flavors">
  <label>Flavors:</label>
  <item>
    <label>Vanilla</label><value>v</value>
  </item>
  <item>
    <label>Strawberry</label><value>s</value>
  </item>
  <item>
    <label>Chocolate</label><value>c</value>
  </item>
</select>
```

You can preselect vanilla and strawberry like this:

```
<instance>
  <data xmlns=""><flavors>v s</flavors></data>
</instance>
```

... initial values for menus

Similarly for the menus example, which looked like this:

```
<select ref="spring">
  <label>Month:</label>
  <item><label>March</label><value>Mar</value></item>
  <item><label>April</label><value>Apr</value></item>
  <item><label>May</label><value>May</value></item>
</select>
```

You can preselect March and April like this:

```
<instance>
  <data xmlns=""><spring>Mar Apr</spring></data>
</instance>
```

... initial values for choices

And for the optgroup example:

```
<select1 ref="drink">
  <label>Drink:</label>
  <item><label>None</label><value>none</value></item>
  <choices>
    <label>Soft drinks</label>
    <item><label>Water</label><value>h2o</value></item>
    <item><label>Milk</label><value>m</value></item>
    <item><label>Juice</label><value>oj</value></item>
  </choices>
  <choices>
    <label>Wine and beer</label>
    <item><label>Red wine</label><value>rw</value></item>
    <item><label>White wine</label><value>ww</value></item>
    <item><label>Beer</label><value>b</value></item>
  </choices>
</select1>
```

... choices

Preselect the value none like this:

```
<instance>
  <data xmlns=""><drink>none</drink></data>
</instance>
```

Hidden Values

- Any values in the instance that haven't been bound to by a control are by definition not visible to the user.
- Therefore there is no need for hidden controls
- To add a hidden value `results` to the search form, we change the instance to:

```
<instance>
  <data xmlns="">
    <q/>
    <results>10</results>
  </data>
</instance>
```

Getting Initial Values From Elsewhere

- You don't have to specify the initial instance in the document itself, because you can load it from an external resource, like this:

```
<instance
  src="http://example.org/templates/t21.xml"/>
```

- The resource then contains your data, like

```
<data><w>640</w><h>480</h><d>8</d></data>
```

- You don't need the `xmlns=""` in external instances, though it doesn't do any harm either.
- You can use a local file, like `src="file:data.xml"`

'Editing' any XML document

- External instances give you immense power
- The `ref` attribute can be any XPath expression
- XPath lets you select any element or attribute in an XML document
- You can bring in *any* XML document as instance, even an XHTML document

... example

- For instance to bind to the `<title>` element in an XHTML document

```
<input ref="h:html/h:head/h:title">...
```

(i.e. the `title` element within the `head` element within the `html` element, all in the XHTML namespace)

- or the `class` attribute on the `body` element:

```
<input ref="h:html/h:body/@class">...
```

- Note the need to put prefixes on elements that are in a namespace. This is why the `xmlns=""` attribute is needed on instance data.

Editing example

Suppose a shop has very unpredictable opening hours (perhaps it depends on the weather), and they want to have a Web page that people can go to to see if it is open. Suppose the page in question has a single paragraph in the body:

```
<p>The shop is <strong>closed</strong> today.</p>
```

Well, rather than teaching the shop staff how to write HTML to update this, we can make a simple form to edit the page instead:

Editing XHTML page

```
<model>
  <instance
    src="http://www.example.com/shop.xhtml"/>
  <submission
    action="http://www.example.com/shop.xhtml"
    method="put" id="change"/>
</model>
...
<select1 ref="/h:html/h:body/h:p/h:strong">
<label>The shop is now:</label>
<item><label>Open</label><value>open</value></item>
<item><label>Closed</label><value>closed</value></item>
</select1>
<submit submission="change"><label>OK</label></submit>
```

- The page *must* be correct XHTML (not HTML)
- The server must accept the "put" method

XPath

XPath selectors look like filename selectors

- **Relative:** "q", "person/name/first"
- **Absolute:** "/data/q"
- **Attributes** of elements with "@": "h:img/@alt"
- Selection is relative to the *context*
- At the top level the context is the instance itself, but some bindings set a new context
- **Current:** .
- **Parent:** ..
- Any descendent: "h:body//h:p"
- There are other selectors, but you seldom need them

... XPath

- A selector selects *all* elements matching the selector.
- In places in XForms where only one is needed, the first is used, eg in `ref="h:body/h:p"`
- There are also methods of specifying *which* you want:
 - `employees/person[1]`
 - `employees/person[position()=last()]`
 - `tutorial[name="xforms"]/tutor`

Submitting

We shall now look at details of submission, like multiple submissions, submission methods, and what happens after submission.

Multiple Submissions

- HTML only allows you to submit the data to one server, in a single way.
- XForms allows you to submit the data to different servers, or in different ways.
- For instance, the search example could allow the user to submit the search string to different search engines:

```
<model>
  <instance><data xmlns=""><q/></data></instance>
  <submission id="com"
    action="http://example.com/search"
    method="get"/>
  <submission id="org"
    action="http://example.org/search"
    method="get"/>
</model>
```

... multiple submissions

and then in the body:

```
<input ref="q"><label>Find:</label></input>
<submit submission="org">
  <label>Search example.org</label>
</submit>
<submit submission="com">
  <label>Search example.com</label>
</submit>
```

Find:

Search example.org Search example.com

Submission Methods

- Just as with HTML there are a number of ways to submit the data.
- In HTML how to submit is expressed in two attributes, `method` and `enctype`
- In XForms it is expressed in `method` only

HTML and XForms Equivalent Submission Methods

HTML	XForms
<code>method="get"</code>	<code>method="get"</code>
<code>method="post"</code> <code>enctype="application/x-www-form-urlencoded"</code>	<code>method="urlencoded-post"</code>
<code>method="post"</code> <code>enctype="multipart/form-data"</code>	<code>method="form-data-post"</code>

... submission methods

- There are some new ways of submission; the most interesting are:
method="post": posts the results as XML
method="put": puts the results as XML.
- An interesting use of this is something like:

```
<submission  
  action="file:results.xml"  
  method="put"/>
```

which saves your results to the local filestore by using the `file:` scheme.

- For a large form, you could have separate 'save to disk' and 'submit' buttons.

Practical: Editing an XML document with XForms

Life after Submit

- The default when values have been submitted is for the result returned by the server to replace the whole document, just as with HTML.
- There are other options, specified with the attribute `replace` on the `submission` element.
- `replace="instance"` replaces only the instance
- `replace="none"` leaves the form document as-is without replacing it.

... example of different submissions

- For instance, for an address-change form for a bank, you can provide two buttons, one to prefill the form with name and address based on the account number, and one to submit the changed results
- The 'find' button replaces the instance with a new instance containing the details of the person with the account number, which you can then change;
- the 'submit' button will then send the changed instance back, leaving the form as-is in the browser to allow further changes or to input a new account number to prefill.

.. example

```
<model>
  <instance><data xmlns="">
    <accountnumber/><name/><address/>
  </data></instance>
  <submission method="get"
    action="http://example.com/prefill"
    id="prefill" replace="instance"/>
  <submission method="get"
    action="http://example.com/change"
    id="change" replace="none"/>
</model>
...
<input ref="accountnumber"><label>Account Number</label></input>
<submit submission="prefill"><label>Find</label></submit>
<input ref="name"><label>Name</label></input>
<textarea ref="address"><label>Address</label></textarea>
<submit submission="change"><label>Submit</label></submit>
```

Controlling Controls

- In HTML you can specify that controls are *disabled*, or *read-only* but the only way you can change the property is with scripting.
- XForms offers easy ways to control these properties, but has other properties you can specify as well

Properties

The 'model binding' properties that you can control are:

- that a value is only **relevant** in certain circumstances (for instance name of spouse only if married)
- that a value is **readonly** in certain circumstances
- that a value is **required** (that a value must be supplied before the form can be submitted)
- that a value has a **constraint** (for instance that the year of birth is earlier than the year of death)
- that the value must conform to a **type** (for instance that it must be an integer), or
- that it is **calculated** from other values (for instance that the total is the sum of some other values).

... properties

Note that in XForms it is the collected *value* that has the property, not the control, but the property shows up on all controls bound to the value.

These properties use a `<bind>` element that goes in the `<model>`. To use `bind`, you must have an explicit `<instance>` element.

Disabled Controls = relevant

To disable controls you use the `relevant` property. For instance, to say that the credit card number only needs to be filled in if the person is paying by credit, you can write:

```
<model>
  <instance><data xmlns="">
    <amount/><method/><cc/><expires/>
  </data></instance>
  <bind nodeset="cc"
    relevant="../method='credit'"/>
  <bind nodeset="expires"
    relevant="../method='credit'"/>
</model>
```

... relevant

- This states that the fields `cc` and `expires` are only relevant when `method` has the value `credit`, and will therefore be disabled for other values of `method`.
- You have to say `"../method"` rather than just `method`, because in a `bind` you are talking about the thing referred to in the `nodeset` (which might be a structured element itself).
- This is an XPath change of context we talked about earlier. It is as if you have done a 'change directory' to that element.
- If you said just `"method"`, it would refer to a child element of `cc` or `expires`.
- You can also use absolute addressing, like `/data/method`, which would have the same effect as `../method` in this case.

... relevant

- A browser is free to decide how disabled controls are presented (and it may also allow you to specify in a stylesheet how they should look), but typically they will be grayed out in the normal way.

... writing the controls

The controls could be written like this (but note that there is no indication that they may get disabled: that is inherited from the value they refer to):

```
<select1 ref="method">
  <label>Method of payment:</label>
  <item>
    <label>Cash</label>
    <value>cash</value>
  </item>
  <item>
    <label>Credit card</label>
    <value>credit</value>
  </item>
</select1>
<input ref="cc"><label>Card number:</label></input>
<input ref="expires"><label>Expiry date:</label></input>
```

... using structured instance values

If we used a structured instance, we could simplify this:

```
<model>
  <instance><data xmlns="">
    <amount/><method/>
    <cc>
      <number/><expires/>
    </cc>
  </data></instance>
  <bind nodeset="cc"
    relevant="../method='credit'"/>
</model>
```

and the controls then reference the children of 'cc':

```
<input ref="cc/number"><label>Card number:</label></input>
<input ref="cc/expires"><label>Expiry date:</label></input>
```

... using grouping on the controls

Instead of:

```
<input ref="cc/number"><label>Card number:</label></input>
<input ref="cc/expires"><label>Expiry date:</label></input>
```

grouping can be used to reset the context of the refs:

```
<group ref="cc">
  <input ref="number"><label>Card number:</label></input>
  <input ref="expires"><label>Expiry date:</label></input>
</group>
```

... works on buttons too

Although putting a ref on a trigger has no effect on the instance value being referred to, the relevance of the value can be used to affect the trigger:

```
<trigger ref="nextok">
  <label>Next</label>
  ...
</trigger>
```

Readonly Controls

Similarly to `relevant`, you can specify a condition under which a value is read-only. For instance:

```
<model>
  <instance><data xmlns="">
    <variant>basic</variant>
    <color>black</color>
  </data></instance>
  <bind nodeset="color"
    readonly="../variant='basic'"/>
</model>
```

This example says that the default value of `color` is `black`, and can't be changed if `variant` has the value `basic`.

Required Controls

A useful new feature in XForms is the ability to state that a value must be supplied before the form is submitted.

The simplest case is just to say that a value is always required. For instance, with the search example:

```
<model>
  <instance><data xmlns=""><q/></data></instance>
  <bind nodeset="q" required="true()" />
  <submission .../>
</model>
```

... required

but like the `readonly` and `relevant` attributes, you can use any XPath expression to make a value conditionally required:

```
<bind nodeset="state"
  required="../country='USA'"/>
```

which says that the value for `state` is required when the value for `country` is "USA".

It is up to the browser to decide how to tell you that a value is required, but it may also allow you to define it in a stylesheet.

Constraint Property

This property allows you to add extra constraints to a value. For instance:

```
<bind nodeset="year" constraint="." &gt; 1970"/>
```

constrains the year to be after 1970.

Note the XPath use of "." to mean "this value".

">" has to be written as > because of XML rules, but you should be used to that already.

Calculate Property

It is possible to indicate that a value in the instance is calculated from other values. For instance:

```
<bind ref="volume"  
  calculate="../height * ../width * ../depth"/>
```

When a value is calculated like this, it automatically becomes `readonly`.

... calculate functions

There are a number of functions available, including:

- arithmetic: + - * div mod
- string manipulation: concat, substring, ...
- date handling: @@@
- booleans: <= < >= > = != and or
- conditionals using 'if':

```
<bind nodeset="taxrate"
  calculate="if(../salary > 50000, 50, 33)"/>
```

Types

- Another useful new feature is the ability to give a value a type. The browser can then check that the values match the required type.
- For instance, if the search example is actually only for searching for numbers (for instance for searching in a bug database), then we only have to add:

```
<bind nodeset="q" type="xsd:integer"/>
```

- This will prevent the value being submitted unless it is an integer.
- You need to add `xmlns:xsd="http://www.w3.org/2001/XMLSchema"` to the root element.

... types

- If you want to collect the URL of someone's homepage, then you can specify

```
<bind nodeset="homepage" type="xsd:anyURI"/>
```

- Some user agents do special things when they know the data type of a value. For instance, when they know that the value is a date, they pop up a date picker rather than require you to type in the characters of the date.

... types

There are a number of useful built-in types you can use, including:

- `xsd:string`, `xsd:normalizedString` (a string with whitespace characters replaced by the space character).
- `xsd:integer`, `xsd:nonPositiveInteger`, `xsd:negativeInteger`, `xsd:nonNegativeInteger`, `xsd:positiveInteger`
- `xsd:boolean`
- `xsd:decimal`, `xsd:double`
- `xsd:date`, `xsd:time`, `xsd:dateTime`
- `xsd:anyURI` (A URI)
- `xforms:listItems` (A space-separated list of strings for use with `select`)
- `xforms:listItem` (A string without any spaces)

Combining Properties

If you have several binds referring to the same value, you can combine them:

```
<bind nodeset="q" type="xsd:integer"
      required="true()" />
```

More than one form in a document

- For more than one form in a document, you can use one model per form, but then you need to identify which form each control refers to
- You do this with an `id` attribute on each model, and a `model` attribute on each control:

... more than one form

```
<model id="search">
  <instance><data xmlns=""><q/></data></instance>
  <submission id="s" .../>
</model>
<model id="login">
  <instance><data xmlns=""><user/><passwd/></data></instance>
  <submission id="l" .../>
</model>
...
<input model="search" ref="q"><label>Find</label></input>
<submit submission="s"><label>Go</label></submit>
...
<input model="login" ref="user"><label>User name</label></input>
<secret model="login" ref="passwd"><label>Password</label></input>
<submit submission="l"><label>Log in</label></submit>
```

More than one instance in a model

- You can have more than one instance in a model.
- You identify which one you want with an id attribute and the use of the instance() function.
- If you don't identify which, then the first instance in the model is used

... more than one instance

```
<model>
  <instance id="currencies">
    <currencies>
      <currency name="USD">125</currency>
      ...
    </instance>
    <instance id="costs">
      <item>
        <date/><amount/><currency/>
        ...
      </item>
    </instance>
  </model>
  ...
  <input ref="instance('costs')/date">
    <label>Date</date>
  </input>
```

... more than one instance

```
<model>
  <instance id="tax" src="/finance/taxes"/>
  <instance>
    <employee xmlns="">
```

Practical: Adapting to earlier answers

Events

- XForms uses a specification called XML Events to deal with eventing
- The important thing to know about XML Events is that it uses the same event mechanism as HTML, only written differently.
- In HTML:

```
<input type="submit"
  onclick="verify(); return true;">
```

says that if the `<input>` element (or any of its children) gets the `click` event, then the piece of code in the `onclick` attribute is performed.

... HTML Events

- We say "or any of its children" because in a case like:

```
<a href="..." onclick="...">A <em>very</em>
  nice place to go</a>
```

you want the `onclick` to be performed even if the click actually happens on the `` element.

- The element that was clicked on is called the **target**
- The element that responds to the event is called an **observer**
- Often target and observer are the same element.
- Three things involved: an event, an observer, and a piece of script (called a **handler**).

Problems with HTML Events

- The event name is hard-wired into the language, rather than being a parameter (so that to be able to deal with a new sort of event you have to add a new attribute)
- You can only use one scripting language (since you can't have two attributes called onclick, one for JavaScript and one for VB)
- The Event names are hardware dependent (e.g. click)
- You are forced to intertwine document and scripting

XML Events

XML Events specifies the relationship between the event, observer and handler in a different way: (HTML example)

```
<input type="button">  
  <script ev:event="DOMActivate"  
    type="text/javascript">  
    DoSomething();  
  </script>  
</input>
```

- The `<script>` element is a **handler** for the **event** `DOMActivate` and in the absence of any other information, the parent element is the **observer** (`<input>` in this case).
- Note that `<script>` elements have to be performed differently from vanilla HTML.

... an advantage

This approach allows you to specify handlers for different scripting languages: (HTML example)

```
<input type="button">
  <script ev:event="DOMActivate"
        type="text/javascript">
    ...
  </script>
  <script ev:event="DOMActivate"
        type="text/vbs">
    ...
  </script>
</input>
```

... another advantage

This approach allows you to specify handlers for different events: (HTML example)

```
<input type="button">
  <script ev:event="event1"
        type="text/javascript">
    ...
  </script>
  <script ev:event="event2"
        type="text/javascript">
    ...
  </script>
</input>
```

Actions

- XForms does not use script, but in-built *actions*.
- You've already seen one example with reset:

```
<trigger>
  <label>Clear all fields</label>
  <reset ev:event="DOMActivate"/>
</trigger>
```

- <reset> is an action that resets all values in the instance to their original values (a copy is made on startup).

Other actions

- setvalue: for setting values in an instance

```
<setvalue ref="total" value="0"/>
```

- send: submit an instance

```
<send submission="s1"/>
```

- message: display a message

```
<message>Done!</message>
```

level="ephemeral": hover style
level="modeless": window style
level="modal": "OK" style

... other actions

- setfocus: set focus on a control

```
<setfocus control="inputdate"/>
```

- action: for grouping

```
<action>  
  <setvalue .../>  
  <setvalue .../>  
</action>
```

- load: load a resource

```
<load resource="doc.html" show="new"/>
```

or

```
<load ref="homeurl" show="..."/>
```

... other actions

- dispatch: dispatch an event

```
<dispatch name="DOMActivate" target="btn1"/>
```

- rebuild, recalculate, revalidate, refresh: almost never needed
- toggle: see later under wizards
- insert, delete, setindex: see later under repeat

Help, hint and alert

All forms controls have, as well as a <label> element, also <help>, <hint> and <alert>.

- help: is displayed if the user asks for help
- hint: is for hover-type hints
- alert: is for information if the value does not validate

```
<input ref="return">
  <label>Return</label>
  <alert>Must be a date later than today</alert>
</input>
```

Events

There are very many events you can catch in XForms, including initialisation events, error notifications, values changing, validity changing, and submission done.

```
<submission id="save"
  action="file:results.xml"
  method="put"
  replace="none">
  <message ev:event="xforms-submit-done">
    Saved!
  </message>
</submission>
...
<submit submission="save">
  <label>Save</label>
</submit>
```

Other ways to specify the event-observer-handler relationship

One way is to move the handler to some other part of the document, and specify the relationship there (like some variants of HTML use the `for` attribute on the `<script>` element):

```
<action ev:observer="#button"
        ev:event="DOMActivate">
  ...
</action>
...
<trigger id="button"/>
```

... another way

Another way is to move the handler somewhere, and specify the relationship in another place with the `<listener>` element:

```
<ev:listener observer="button"
             handler="dosomething"
             event="DOMActivate"/>
...
<action id="dosomething">...</action>
...
<input type="submit" id="button"/>
```

- This allows you to use the same handler for more than one observer
- Note that the `ev:` prefix goes on the element in this case, not the attributes.

... another way

And finally, you can specify the relationship on the observer itself:

```
<action id="dosomething">
  ...
</action>
...
<trigger ev:handler="dosomething"
  ev:event="DOMActivate"/>
```

Wizards: toggle and switch

These are used to reveal and hide parts of the interface.

```
<switch>
  <case id="inputname">
    <input ref="name">...</input>
    <trigger>
      <label>Next</label>
      <toggle case="inputage"
        ev:event="DOMActivate" />
    </trigger>
  </case>
  <case id="inputage">
    <input ref="age">...</input>
    <trigger>...</trigger>
  </case>
  ...
</switch>
```

Repeat

Repeat allows you to bind to repeating items in an instance

```
<shoppinglist>
  <buy>eggs</buy>
  <buy>milk</buy>
  ...
</shoppinglist>
```

```
<repeat ref="buy" id="shoprepeat">
  <input ref="."><label>Buy</label></input>
</repeat>
```

A repeat sets the XPath context.

insert, delete, setfocus

You can use these with `<repeat>` to add and delete items, and to focus on a specific item.

```
<trigger>
  <label>Add</label>
  <insert ev:event="DOMActivate"
    nodeset="buy"
    at="index('shoprepeat') "
    position="after"/>
</trigger>
```

```
<trigger>
  <label>Delete</label>
  <delete ev:event="DOMActivate"
    nodeset="buy"
    at="index('shoprepeat') "/>
</trigger>
```

Implementations

- At release XForms had more implementations announced than any other W3C spec had ever had at that stage
- Different types of implementation:
 - plugin
 - native
 - 'zero install'
 - proxy
- Many big players doing implementations, e.g.
 - Novell
 - Oracle
 - IBM
 - Sun

"The age of the fat client is past" -- an implementor

The Future

Experience with XForms 1.0 has revealed a number of things:

- Some ambiguities
- Some missing functionality
- Some 'low hanging fruit': additional features implemented on several implementations, but in different ways

A future iteration of XForms will address these issues.

More Information

The origin: www.w3.org/Markup/Forms, and if your company is a member:
www.w3.org/Markup/Forms/Group

XForms: <http://www.w3.org/TR/xforms/>

XPath: <http://www.w3.org/TR/xpath>

XPath quick reference: <http://www.mulberrytech.com/quickref/XSLTquickref.pdf>

XML Events: <http://www.w3.org/TR/xml-events/>

Practical: shopping basket style