XForms 2.0: What's new?

Steven Pemberton, CWI, Amsterdam

Abstract

XForms is a declarative language for defining applications on the web and elsewhere, used worldwide for large and small applications. This paper gives an overview of what has changed between the previous version and the up-coming XForms 2.0.

Introduction

XForms was originally designed as a new XML-based markup language for forms on the web, and version 1.0 [XF1] was just that. However, after initial experience, it was realised that the design had followed HTML too slavishly, in particular by only accepting static strings in places where expressions would be more useful. With some generalisation XForms became more powerful in the shape of XForms 1.1 [XF11], and in the process became a Turing-complete language that could still do forms, but very much more as well. Rather than procedural or functional, XForms's programming model is declarative, where you define *what* you want to achieve rather than exactly *how*. This has proven to be very successful in reducing the time and costs of producing applications, typically by a factor of ten, over a wide range of different applications, both large and small.

XForms 2.0 [XF2] continues the process of generalisation, making the definition of applications even easier. We describe here some of the major differences.

XForms

The processing model of XForms is based on state: there is data, and a description of that data, such as types, constraints, and relationships between values. Initially the system ensures that the values are consistent with the description, and then goes into stasis until something changes, either from an action by the user, or internally from the system. Normally the system returns the system to stasis automatically, using an expression recalculation algorithm similar to how spreadsheets work, but the system also dispatches events to signal state change, which event listeners can respond to with actions, which in turn can cause further state changes.

Thus we split the description below into data and its descriptions and types, expressions, controls for user interaction, events, and actions. Finally submission is how XForms communicates with the outside world, talking to servers.

Data

Earlier version of XForms only accepted XML as a data format; XForms 2.0 extends this by defining mappings from external formats to an internal XML model, thus making those external formats appear as XML internally, but still round-tripped to the original formats. Mappings are currently defined for JSON and CSV, but it is possible for implementations to add other formats.

As an example, a JSON object such as

```
{"father": {"given": "Mark", "family": "Smith"}, "mother": {"given": "Mary", "family": "Smith"}}
```

would be transformed to the equivalent of the following:

```
<json type="object">
   <father type="object">
      <given>Mark</given>
      <family>Smith</family>
   </father>
   <mother type="object">
      <given>Mary</given>
      <family>Smith</family>
   </mother>
</json>
```

and a value would be selected with an XPath expression such as `mother/given`.

An array such as

```
{"load": [0.31, 0.33, 0.32]}
```

is transformed to

```
<json type="object">
  <load type="array">
    <_ type="number">0.31</_>
    <_ type="number">0.33</_>
    <_ type="number">0.32</_>
  </load>
</json>
```

and accessed with an expression such as `load/_[1]`.

Data Properties

Model Item Properties (MIPs) are properties of data nodes within XForms: type, constraints on the value, the relationship to other values, whether the data is read-only or not, and under what conditions the data is relevant and required. Originally a MIP for a data node had to be defined in one go in a single `bind` element. Now it may be extended over several binds, with rules for how they are combined.

The new `whitespace` MIP defines how whitespace is treated in values when a user inputs a value: trimmed, compressed or removed entirely. For instance, if the user inputs spaces when entering a credit card number, they can be automatically and silently removed.

See also below on labels.

Types

There are a number of new types:

- two IRI types: These allow the use of, and check the syntax of, 'international' web addresses and the likes, such as `https://zh.wikipedia.org/wiki/Wikipedia:关于中文维基百科/en`
- iemail type: allowing and checking 'international' email addresses such as 甲斐@黒川.日本. Along with this, the existing 'legacy' email type has been improved: the original definition was rather loose; this has been tightened up to more closely match, and check, real-world email addresses.
- Telephone number type: allowing the checking of telephone numbers, such as +1 (234) 1234567
- HTMLFragment type: this is a marker type that signals the intention of what the content represents, but doesn't check its validity. It exists primarily to allow rich-text editing controls to bind to values of such type.

The definition of validity has been slightly improved by allowing non-required values to be considered valid if empty. This simplifies the definition of controls for optional values. Consequently, the standard XForms datatypes no longer necessarily include empty as a possible value.

Expressions

XForms 2.0 uses XPath 2 [XP2], instead of XPath 1 [XP1]; this allows amongst other things atomic values and typed values; however, the expression language part of XForms 2.0 has been defined as a separate module allowing future versions of XForms easily to switch to newer versions of XPath [XP3].

Earlier versions of XForms only allowed data values to be injected into element content of controls (using the `output` control). With the addition of Attribute Value Templates (AVTs), calculated values can now also be injected into attributes, enabling the deprecation of many child elements.

For instance, where you previously had to say:

```
<output ref="maptile">
   <mediatype value="site/type"/>
</output>
```

you can now write:

```
<output ref="maptile" mediatype="{site/type}"/>
```

AVTs also create many new presentation options; for instance `class` values can now be calculated dynamically:

```
<output ref="balance" class="{if(. &lt; 0, 'negative', 'positive')}"/>
```

Custom functions and variables have been added.

New functions are available:

- bind(): returns the nodeset defined in a `bind` element, allowing expressions to be defined that work independently of how the data is structured.
- valid(), relevant(), readonly(), required(): give access to MIP values.
- case(): for identifying which `case` of a `switch` has been selected.
- serialize(): serializes a value to a string in the same way a submission would work.
- parse(): the complement of `serialize` - turns a string into an XML document element node, in the same way an external instance would be read.
- element(), attribute(): allows the construction of elements from values.
- eval(), eval-in-context(): allows the evaluation of a string as an expression.
- location-uri(), location-param(): functions to reveal how the XForm was called (i.e., with what URL).
- uri-scheme(), uri-scheme-specific-part(), uri-authority(), uri-user-info(), uri-host(), uri-port(), uri-path(), uri-query(), uri-fragment(), uri-param-names(), uri-param-values(): helper functions for extracting parts of a URI. These can be used in initialising data values based on how the XForm was called.

Controls

Earlier versions of XForms distinguished whether a control expected a single value, such as `<input ref="height">` or a sequence of values, such as `<repeat nodeset="person">`. However, it was observed that users did not benefit from this distinction, so the `nodeset` attribute has now been deprecated in favour of `ref` everywhere. If a control expects a single value, and gets a sequence, it selects the first in the sequence (which is what also happened in earlier versions).

Most controls have the helper elements `label`, `hint`, `help`, `alert` as sub-elements. For example:

```
<input ref="age">
   <label>Age</label>
   <alert>Must be a non-negative whole number</alert>
</input>
```

They may now alternatively be defined as attributes:

```
<input ref="age" label="Age" alert="Must be a non-negative whole number"/>
```

but also they may be attached to the node itself as a MIP, with a `bind`:

```
<bind ref="age" type="nonNegativeInteger"
      label="Age" alert="Must be a non-negative whole number"/>
 ...
<input ref="age"/>
```

This has, amongst other things, the advantage of allowing the condition and matching message to be kept close to each other:

```
<bind ref="age" label="Age">
   <bind type="integer" alert="Must be a whole number"/>
   <bind constraint=". &gt;= 0 and . &lt;= 120" alert="Must be between 0 and 120"/>
</bind>
```

The `label` elements and attributes are consequently now optional on controls.

The semantics of how `label`, `help`, `hint`, and `alert` work have been unified with the `output` control.

Previously if you wanted to know which iteration of a repeat was being processed, there was a function `index` to tell you. However, this interacted poorly in combination with the XForms expression evaluation algorithm, since a change to the index didn't initiate a recalculation. Repeat controls now have an optional `indexref` attribute which binds to an instance value that is kept in sync with the repeat index, thus allowing automatically updated calculations to take account of such values. Similarly with `switch` controls, which now have an optional `caseref` attribute. These attributes work bi-directionally: for instance you can use `caseref` to determine which `case` has been selected, but also write to it to cause a particular `case` to be selected. This means that actions such as `<setindex/>` are now no longer needed (though are still available for backwards compatibility).

The repeat and group elements now have an `appearance='minimal'` attribute to turn them from block elements into inline elements. While this was already possible with CSS definitions, this shorthand makes a frequently occurring use case easier to define.

There is a new `dialog` element for describing simple dialogues.

Actions and Events

All actions now have an `iterate` attribute to allow them to iterate over a range of values:

```
<setvalue iterate="item[@selected=true()]" ref="." value=". + 1"/>
```

The XForms recalculation algorithm is described in four steps, and to match this, in previous versions of XForms there were four actions: `rebuild, recalculate, revalidate, and refresh` ; however, it turns out that all most users want to ensure is that everything is up-to-date A new action `<update/>` subsumes the previous actions: . The `update` action worries about the details.

It is now possible to call script (such as Javascript) from actions; this includes passing parameters.

In all versions of XForms the state of all instances is saved at start-up to enable the `reset` action. Now it is possible to define at which point the instances are saved with the `retain` action, and also to reset one or more instances rather than all of them.

It is now possible to add properties to dispatched events. Event handlers are no longer disabled when their associated controls are disabled.

Submission

It is now possible to override the mediatype of an incoming document. This is useful for cases where the server uses the wrong, or a too-broad, mediatype, and for cases where the mediatype doesn't expose that the document is an application of XML.

The default submission method is `get` (previously there was no default).

A new submission attribute `nonrelevant` replaces the existing `relevant`, adding new options (`relevant` was a boolean, but there are now more than two options for relevance processing).

Document Structure

XForms has always been designed to be integrated in another markup language, whether XHTML, SVG, ODF, or whatever. To that end, XForms has not needed a root element. However, for those occasions that an implementation needs a free-standing usage of XForms, there is now an optional root element `<form>`.

Conclusion

Updating an existing language has the extra problem that you have to deal with legacy: you want to make changes without invalidating existing documents. This sometimes blocks the most obvious change, and demands creativity to find a suitable expression of the required functionality.

XForms 2.0 is a surprisingly expressive language. Declarative programming requires a different mindset, but once acquired allows the definition of apllications in an unexpectedly small number of lines of code. Examples include maps [Maps], multi-lingual applications and a presentation program [FC], and several games [Game1, Game2].

References

[XF1] Micah Dubinko, *et al.*, (eds.), *XForms 1.0*, W3C, 2003, https://www.w3.org/TR/2003/REC-xforms-20031014/

[XF11] John M. Boyer, (ed.), *XForms 1.1*, W3C, 2009, https://www.w3.org/TR/2009/REC-xforms-20091020/

[XF2] E. Bruchez, *et al.*, (eds.), *XForms 2.0*, W3C, 2018, https://www.w3.org/community/xformsusers/wiki/XForms_2.0

[XP2] Anders Berglund, *et al*, (eds.), *XML Path Language (XPath) 2.0 (Second Edition)*, W3C, 2010, https://www.w3.org/TR/xpath20/

[XP3] Jonathan Robie, *et al.*, (eds.), *XML Path Language (XPath) 3.1*, W3C, 2017, https://www.w3.org/TR/xpath-3/

[Maps] Steven Pemberton, *XForms: an Unusual Worked Example*, CWI, 2015, https://homepages.cwi.nl/~steven/Talks/2015/11-05-example/

[FC] Steven Pemberton, *Form, and Content: Data-Driven Forms*, XML Prague 2018 Conference Proceedings, pp 213-28, http://archive.xmlprague.cz/2018/files/xmlprague-2018-proceedings.pdf#page=225

[Game1] Steven Pemberton, *A Game in XForms*, CWI, 2017, https://homepages.cwi.nl/~steven/Talks/2017/06-10-iot/game-demo.html

[Game2] Steven Pemberton, *Minesweeper in XForms*, CWI, 2018, https://homepages.cwi.nl/~steven/Talks/2018/02-10-prague/minesweeper.xml