XForms 1.0

W3C Working Draft 18 January 2002

This version:
http://www.w3.org/TR/2002/WD-xforms-20020118 (single HTML file, diff-marked HTML, PDF, Zip archive)
Latest version:
http://www.w3.org/TR/xforms/
Previous versions:
http://www.w3.org/TR/2001/WD-xforms-20011207

Editors:
Micah Dubinko , Cardiff <mdubinko@Cardiff.com>
Josef Dietl , Mozquito Technologies <josef@mozquito.com>
Leigh L. Klotz, Jr. , Xerox Corporation <Leigh.Klotz@pahv.xerox.com>
Roland Merrick , IBM <Roland_Merrick@uk.ibm.com>
T. V. Raman , IBM <tvraman@almaden.ibm.com>

Abstract

XForms is an XML application that represents the next generation of Forms for the Web. By splitting traditional XHTML forms into three parts - data model, instance data, and user interface - it separates presentation from content, allows reuse, gives strong typing - reducing the number of round-trips to the server, as well as offering device independence and a reduced need for scripting.

XForms is not a free-standing document type, but is intended to be integrated into other markup languages, such as XHTML.

Status of this Document

Last Update: $Date: 2002/01/16 23:32:13 $

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This is a W3C Last Call Working Draft of the XForms 1.0 specification, for review by W3C members and other interested parties. The Last Call review period ends on 22 February 2002 at 2359Z. Please send review comments before the end of the review period to www-forms-editor@w3.org. This list is archived at http://lists.w3.org/Archives/Public/www-forms-editor/.

Following completion of Last Call, the XForms Working Group has agreed to advance the specification according to the following exit criteria:

1. Sufficient reports of implementation experience have been gathered to demonstrate that XForms processors based on the specification are implementable and have compatible behavior.
2. An implementation report shows that there is at least one implementation of each feature.

3. Formal responses to all comments received by the Working Group.

If these criteria are met, the specification will advance to Proposed Recommendation, otherwise the specification will enter a Candidate Recommendation phase to ensure that the above criteria are met.

This document is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of current public W3C Working Drafts can be found at http://www.w3.org/TR.

This document has been produced as part of the W3C HTML Activity.

Please send detailed comments on this document to www-forms@w3.org, the public forum for discussion of the W3C's work on web forms. To subscribe, send an email to the above address with the word *subscribe* in the subject line (include the word *unsubscribe* if you want to unsubscribe). The archive for the list is accessible online.

Table of Contents

Appendices

1 About the XForms 1.0 Specification

1.1 Background

Forms are an important part of the Web, and they continue to be the primary means for enabling interactive web applications. Web applications and electronic commerce solutions have sparked the demand for better web forms with richer interactions. XForms are the response to this demand and provide a new platform-independent markup language for online interaction between an XForms Processor and a remote user agent. XForms are the successor to HTML forms, and benefit from the lessons learned from HTML forms.

Further background information on XForms can be found at http://www.w3.org/MarkUp/Forms.

1.2 Reading the Specification

This specification has been written with various types of readers in mind—in particular XForms authors and XForms implementors. We hope the specification will provide authors with the tools they need to write efficient, attractive and accessible documents without overexposing them to the XForms implementation details. Implementors, however, should find all they need to build conforming XForms Processors. The specification begins with a general presentation of XForms before specifying the technical details of the various XForms components.

The specification has been written with various modes of presentation in mind. In case of a discrepancy, the online electronic version is considered the authoritative version of the document.

This document uses the terms **may**, **must**, and **should** in accord with RFC 2119.

1.3 How the Specification is Organized

The specification is organized into the following chapters:

Chapters 1 and 2

An introduction to XForms. The introduction outlines the design principles and includes a brief tutorial on XForms.

Chapters 3 and up

XForms reference manual. The bulk of the reference manual consists of the specification of XForms. This reference defines XForms and how XForms Processors must interpret the various components in order to claim conformance.

Appendixes

Appendixes contain a normative description of XForms described in XML Schema, information on references, and other useful information.

1.4 Documentation Conventions

Throughout this document, the following namespace prefixes and corresponding namespace identifiers are used:

xforms: The XForms namespace **3.1 The XForms Namespace**
html: The XHTML namespace [XHTML 1.0]
xsd: The XML Schema namespace [XML Schema part 1]
xsi: The XML Schema for instances namespace [XML Schema part 1]
xlink: The XLink namespace [XLink]
ev: The XML Events namespace [XML Events]
my: Any user defined namespace

This is only a convention; any namespace prefix may be used in practice.

The following typographical conventions are used to present technical material in this document.

Official terms are defined in the following manner: [Definition: You can find most **terms** in chapter **12 Glossary Of Terms**]. Links to **term**s may be specially highlighted where necessary.

The XML representations of various elements within XForms are presented as follows: Listed are the element name, names of all attributes, allowed values of attributes appearing after a "=" character, default values of attributes appearing after a ":" character, and allowed content. One or more headings below the listing provide additional explanatory information.

Example: XML Syntax Representation `<example>`

```
<example
  count = xsd:integer
  size = (small | medium | large) : medium
>
  <!-- Content: (allowed-content) -->
</example>
```

count - description of this attribute
size - description of this attribute

Certain common attributes **3.2 Horizontally Applicable Markup** are not shown in the syntax representations except when special attention needs to be called to their presence.

Examples are set off typographically:

Example: Example item

```
Example Item
```

References to external documents appear as follows: [Sample Reference] with links to the references section of this document.

Sample Reference
Reference - linked to from above.

The following typesetting convention is used for non-normative commentary:

Note:

A gentle explanation or admonition to readers.

Issue (issue-id):
Issue-Name

A specific issue to which input from readers is requested, not intended for final publication.

For diff-marked formatted text, note that newly added text appears like this, changed text appears like this, and deleted text appears like this.

2 Introduction to XForms

This chapter provides an easily approachable description of XForms. Not every feature of XForms is covered here. For a complete and normative description of XForms, refer to the remainder of this document. The following subsections develop a complete example of an XForms application that is hosted in an XHTML document. The complete example is found in **E.1 XForms In XHTML**.

2.1 Separating Purpose From Presentation

A typical form starts off with a *purpose*, e.g., data collection. This purpose is realized by creating an interactive *presentation* that allows the user to provide the requisite information. The resulting *data* is the result of completing the form.

HTML forms failed to separate the purpose of a form from its presentation; additionally, they only offered a restricted representation for data captured through the form. Here is a summary of the primary benefits of using XForms:

Strong typing

Submitted data is strongly typed and can be checked using off-the-shelf tools. Type validation rules help client-side validation, and such validation code can be automatically generated.

Existing schema re-use

This obviates duplication, and ensures that updating the validation rules as a result of a change in the underlying business logic does not require re-authoring validation constraints within the XForms application.

External schema augmentation

This enables the XForms author go beyond the basic set of constraints available from the back-end. Providing such additional constraints as part of the XForms Model enhances the overall usability of the resulting web application.

XML submission

This obviates the need for custom server-side logic to marshal the submitted data to the application back-end. The received XML instance document can be directly validated and processed by the application back-end.

Internationalization

Using XML 1.0 for instance data ensures that the submitted data is internationalization ready.

Enhanced accessibility

XForms separates content and presentation. User interface controls encapsulate all relevant metadata such as labels, thereby enhancing accessibility of the application when using different modalities. XForms user interface controls are generic and suited for device-independence.

Multiple device support

The high-level nature of the user interface controls, and the consequent intent-based authoring of the user interface makes it possible to re-target the user interaction to different devices.

Declarative event handlers

By defining XML-based declarative event handlers such as `setFocus`, `message`, and `setValue` that cover common use cases, the majority of XForms documents can be statically analyzed; contrast this with the present practice of using imperative scripts for event handlers.

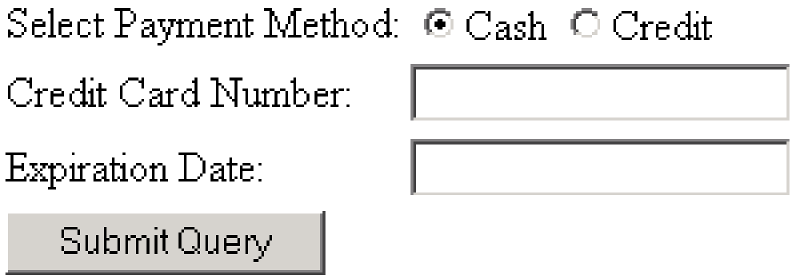
2.2 Current Approach: HTML

Consider a simple electronic commerce form authored in HTML:

Example: HTML Form

```
<html>
  <head>
    <title>eCommerce Form</title>
  </head>
  <body>
    <form action="http://example.com/submit" method="post">
      <table summary="Payment method selector">
      <tr>
      <td><p>Select Payment Method:</p></td>
      <td><label><input type="radio" name="as" value="cash"/>Cash</label>
      <label><input type="radio" name="as" value="credit"/>Credit</label></td>
      </tr>
      <tr>
      <td><label for="cc">Credit Card Number:</label></td>
      <td><input type="text" name="cc" id="cc"/></td>
      </tr>
      <tr>
      <td><label for="exp">Expiration Date:</label></td>
      <td><input type="text" name="exp" id="exp"/></td>
      </tr>
      <tr>
      <td colspan="2"><input type="submit"/></td>
      </tr>
      </table>
    </form>
  </body>
</html>
```

A user agent might render this form as follows:



Select Payment Method: ● Cash ○ Credit

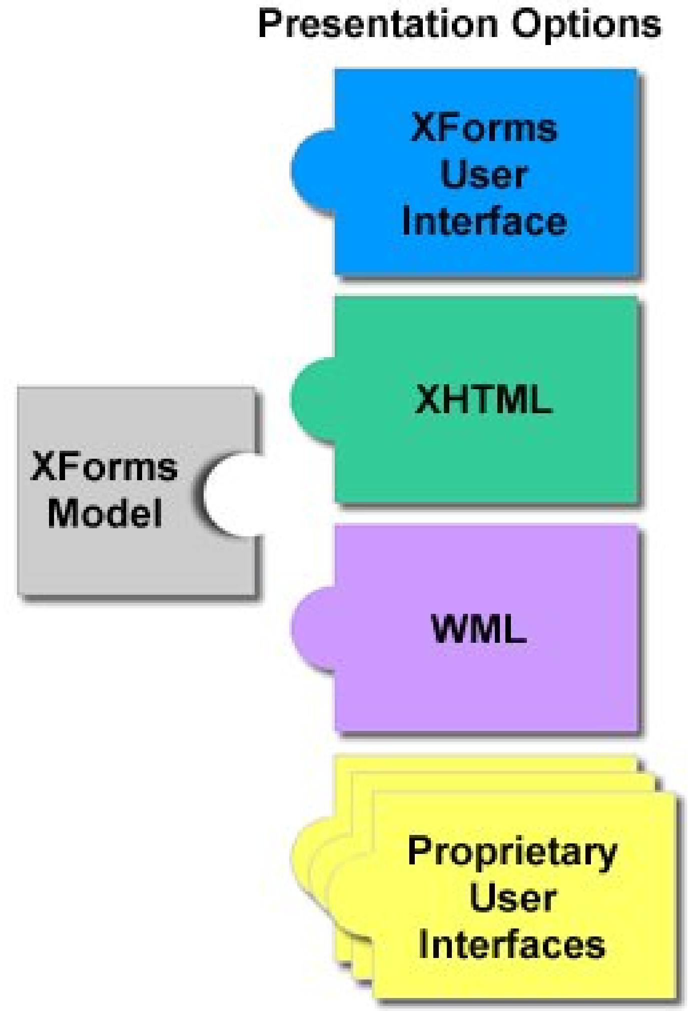
Credit Card Number:

Expiration Date:

This form makes no effort to separate purpose (data collection semantics) from presentation (the `input` form controls), and offers no control over the pair serialization of the resulting data as name-value pairs. In contrast, XForms greatly improve the expressive capabilities of electronic forms.

2.3 Transition to XForms

In the XForms approach, forms are comprised of a section that describes what the form does, called the XForms Model, and another section that describes how the form is to be presented. XForms 1.0 defines the XForms User Interface, which is a device-independent, platform-neutral set of form controls suitable for general-purpose use. The user interface is *bound* to the XForms model via the XForms binding mechanism; This flexible architecture allows others to attach user interfaces to an XForms Model as illustrated here:



The simplest case involves authoring the new XForms form controls, leaving out the other sections of the form. To convert the previous form into XForms this way, a `model` element is needed in the `head` section of the document:

```
<xforms:model>
  <xforms:submitInfo action="http://examples.com/submit" id="submit"/>
</xforms:model>
```

With these changes to the containing document, the previous example could be rewritten like this (note that we have intentionally defaulted the XForms namespace prefix in this example):

```
<selectOne ref="as">
  <caption>Select Payment Method</caption>
  <choices>
    <item>
      <caption>Cash</caption>
      <value>cash</value>
    </item>
    <item>
      <caption>Credit</caption>
      <value>credit</value>
    </item>
  </choices>
</selectOne>

<input ref="cc">
  <caption>Credit Card Number</caption>
</input>

<input ref="exp">
  <caption>Expiration Date</caption>
</input>

<submit submitInfo="submit">
  <caption>Submit</caption>
</submit>
```

Notice the following features of this design:

- The user interface is not hard-coded to use radio buttons. Different devices (such as a voice browser) can render the concept of "selectOne" as appropriate.
- Form controls always have captions directly associated with them, as child elements—this is a key feature designed to enhance accessibility.
- There is no need for an enclosing `form` element, as in HTML. See (See **2.6 Multiple Forms per Document** for details on how to author multiple forms per document)
- Markup for specifying form controls has been simplified
- Data gets submitted as XML.

With these changes, the XForms Processor will be able to directly submit XML instance data. The XML is constructed by creating a root element with child elements reflecting the names specified in each form control via attribute `ref`. In this example, the submitted data would look like this:

```
<instanceData>
  <as>Credit</as>
  <cc>1235467789012345</cc>
  <exp>2001-08</exp>
</instanceData>
```

2.4 Providing XML Instance Data

XForms processing keeps track of the state of the partially filled form through instance data. Initial values for the instance may be provided via element `instance`. Element `instance` holds a skeleton XML document that gets updated as the user fills out the form. Element `instance` gives the author full control on the structure of the submitted XML data, including namespace information. When the form is submitted, the instance data is serialized as an XML document. The initial instance data is defined in the `instance` element inside the `model` element, as follows:

```
<xforms:model>
  <xforms:instance>
    <payment as="credit" xmlns="http://commerce.example.com/payment">
      <cc/>
      <exp/>
    </payment>
  </xforms:instance>
  <xforms:submitInfo action="http://example.com/submit" method="post"/>
</xforms:model>
```

This design has features worth calling out:

- There is complete flexibility in the structure of the XML. Notice that XML namespaces are now used, and that a wrapper element of the author's choosing contains the instance data.

- Empty elements `cc` and `exp` serve as place-holders in the XML structure, and will be filled in with form data provided by the user.
- An initial value (`"credit"`) for the form control is provided through the instance data, in this case an attribute `as`. In the submitted XML, this initial value will be replaced by the user input.

To connect this instance data with form controls, the `ref` attributes on the form controls need to point to the proper part of the instance data, using binding expressions.

Example: Binding Expression

```
 xmlns:my="http://commerce.example.com/payment"...
  <xforms:selectOne ref="my:payment/@as">
    ...
  <xforms:input ref="my:payment/my:cc">
    ...
  <xforms:input ref="my:payment/my:exp">
```

Binding expressions are based on XPath [XPath 1.0], including the use of the `@` character to refer to attributes, as seen here.

2.5 Constraining Values

XForms allows data to be checked for validity as the form is being filled. Referring to the earlier HTML form in **2.2 Current Approach: HTML**, there are several desirable aspects that would only be possible to ensure through the addition of unstructured script code:

- The credit card information form controls `cc` and `exp` are only relevant if the "credit" option is chosen in the `as` form control.
- The credit card information form controls `cc` and `exp` should be required when the "credit" option is chosen in the `as` form control.
- The form control `cc` should accept digits only, and should have between 14 and 18 digits.
- The form control `exp` should accept only valid month/date combinations.

By specifying an additional component, model item constraints, authors can include rich declarative validation information in forms. Such information can be taken from XML Schemas as well as XForms-specific constraints, such as `relevant`. XForms constraints appear on `bind` elements, while Schema constraints are expressed in an XML Schema fragment, either inline or external. For example:

```
... xmlns:my="http://commerce.example.com/payment"...
<xforms:bind ref="my:payment/my:cc"
    relevant="../my:payment/@as = 'credit'"
    required="true"
    type="my:cc"/>

<xforms:bind ref="my:payment/my:exp"
    relevant="../my:payment/@as = 'credit'"
    required="true"
    type="xsd:gYearMonth"/>

<xforms:schema>
  <xsd:schema ...>
    ...
    <xsd:simpleType name="cc">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{14,18}"/>
      </xsd:restriction>
    </xsd:simpleType>
    ...
  </xsd:schema>
</xforms:schema>
```

2.6 Multiple Forms per Document

XForms processing places no limits on the number of individual forms that can be placed in a single containing document. When a single document contains multiple forms, each form needs a separate `model` element. The first `model` element may omit a unique `id` attribute (as have all the examples above), but subsequent `model` elements require an `id` so that they can be referenced from elsewhere in the containing document.

In addition, form controls need to specify the `model` element contains the instance data to which they bind. This is accomplished through a `model` attribute alongside the `ref` attribute. The default for the `model` attribute is the first `model` element in document order.

The next example adds an opinion poll to our electronic commerce form.

```
<xforms:model>
  <xforms:instance>
    ...payment instance data...
  </xforms:instance>
  <xforms:submitInfo action="http://example.com/submit" method="post"/>
</xforms:model>

<xforms:model id="poll">
  <xforms:submitInfo .../>
</xforms:model>
```

Additionally, the following markup would appear in the body section of the document:

```
<xforms:selectOne ref="pollOption" model="poll">
  <xforms:caption>How useful is this page to you?</xforms:caption>
  <xforms:choices>
    <xforms:item>
      <xforms:caption>Not at all helpful</xforms:caption>
      <xforms:value>0</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:caption>Barely helpful</xforms:caption>
      <xforms:value>1</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:caption>Somewhat helpful</xforms:caption>
      <xforms:value>2</xforms:value>
    </xforms:item>
    <xforms:item>
      <xforms:caption>Very helpful</xforms:caption>
      <xforms:value>3</xforms:value>
    </xforms:item>
  </xforms:choices>
</xforms:selectOne>

<xforms:submit submitInfo="poll">
  <xforms:caption>Submit</xforms:caption>
</xforms:submit>
```

The main difference here is the use of `model="poll"`, which identifies the instance.

Note that complete examples can be found in **E Complete XForms Examples**

3 Document Structure

The XForms specification is an application of XML [XML 1.0] and has been designed for use within other XML vocabularies—in particular within XHTML [XHTML 1.0]. This chapter discusses the structure of XForms that allow this specification to be used with other document types.

3.1 The XForms Namespace

The XForms namespace has the URI: `http://www.w3.org/2002/01/xforms`. Any future Working Drafts are expected to use a different identifier, though a final identifier will be allocated before XForms becomes a W3C Recommendation.

XForms Processors must use the XML namespaces mechanism [XML Names] to recognize elements and attributes from this namespace.

3.2 Horizontally Applicable Markup

Every element defined in this specification declares attribute `id` of type `xsd:ID`—see the schema for XForms—this allows these elements to be referenced via attributes of type `xsd:idref`.

Foreign-namespaced attributes are allowed on any XForms element. The XForms Processor must ignore any foreign-namespaced elements or attributes that are unrecognized.

Note that except where specifically allowed by the Schema for XForms, foreign-namespaced elements are not allowed as content of elements in the XForms namespace.

3.3 Model

This section describes XForms element `model` used as a container for XForms elements defining the XForms model. The containing document may contain one or more `model` elements. Element `model` defines the underlying model to which the XForms document *binds* user interaction. Hence, `model` elements occur before the user interaction markup. The content of element `model` is typically not rendered. As an example, `model` elements occur within element `html:head` of an XHTML document, whereas XForms user interface markup appears within element `html:body`.

Example: XML Representation: <model>

```
<model
  extensionFunctions = list of QNames
>
  <!-- Content: instance?, schema?, (privacy|submitInfo|bind|action|extension)* -->
</model>
```

extensionFunctions - Optional list of XPath extension functions used by this XForms Model. It is an error to use an undeclared extension function.

Element `model` can contain the following elements.

Instance
Defines skeleton instance document and holds initialization data if any—see **3.4 instance**.

Schema
Defines schema for the instance—see **3.5 schema**

submitInfo
Holds submit details—see **3.6 submitInfo**

Bindings
Elements `bind` that establish one or more XForms bindings to define model item constraints—see **6 Constraints**.

Privacy
Establishes P3P properties—see **3.7 privacy**

Action
Event handlers—see **10.16 action**. This allows the author to handle events that arrive at node `model`—see the processing model **4 Processing Model**.

Extension
Extension elements if any—see **8.12.4.5 extension**

Example: Model

```
<model xmlns="http://www.w3.org/2002/01/xforms" id="Person">
   <instance xlink:href="http://example.com/cgi-bin/get-instance" />
   <schema xlink:href="Schema-Questionnaire.xsd" />
   ...
</model>
```

3.4 instance

Element `instance` contains a skeleton instance document that provides initial instance data. The instance data may be defined inline or obtained from an external URI.

Example: XML Representation: `<instance>`

```
<instance xlink:href = xsd:anyURI >
  <!-- Content: (##any) -->
</instance>
```

xlink:href - Optional link to externally defined instance data

The content of the `instance` element is arbitrary XML in any namespace. other than the XForms namespace. The content of this element is treated as opaque data. Authors must ensure that proper namespace declarations are used for content within the `instance` element.

3.5 schema

Element `schema` contains the schema defining the instance. The schema may be defined inline or obtained from an external URI.

Example: XML Representation: `<schema>`

```
<schema xlink:href = xsd:anyURI >
  <!-- Content: ##other (though typically <xsd:schema>) -->
</schema>
```

xlink:href - Optional link to an externally defined schema.

3.6 submitInfo

Element `submitInfo` encodes how, where and what to submit.

Example: XML Representation: `<submitInfo>`

```
<submitInfo
  (single node binding attributes)
  action = xsd:anyURI
  mediaTypeExtension = "none" | qname-but-not-ncname : "none"
  method = "post" | "get" | qname-but-not-ncname : "post"
  version = xsd:NMTOKEN
  indent = xsd:boolean
  encoding = xsd:string
  mediaType = xsd:string
  omitXMLDeclaration = xsd:boolean
  standalone = xsd:boolean
  CDATASectionElements = list of xsd:QName
  replace = "all" | "instance" | "none" | qname-but-not-ncname : "all"
>
  <!-- Content: XForms Actions -->
</submitInfo>
```

single node binding attributes - optional selector enabling submission of a portion of the instance data
action - Required destination for submitting instance data.
mediaTypeExtension - Optional information describing the serialization format. This is in addition to the mediaType.
method - Optional indicator as to the protocol to be used to transmit the serialized instance data.
version - corresponds to the `version` attribute of `xsl:output`
indent - corresponds to the `indent` attribute of `xsl:output`
encoding - corresponds to the `encoding` attribute of `xsl:output`
mediaType - corresponds to the `media-type` attribute of `xsl:output`
omitXMLDeclaration - corresponds to the `omit-xml-declaration` attribute of `xsl:output`
standalone - corresponds to the `standalone` attribute of `xsl:output`
CDATASectionElements - corresponds to the `cdata-section-elements` attribute of `xsl:output`
replace - specifier for how the information returned after submit should be applied.

Note:

Many of these attributes correspond to XSLT attributes [XSLT]. Note that the XSLT attributes `doctype-system` and `doctype-public` are not supported in XForms processing.

Note:

Note also that attribute `mediaTypeExtension` is useful in cases where a media type alone is not sufficiently precise. For instance, a SOAP envelope would not be adequately described simply by "text/xml", additional information would be required.

3.7 privacy

Element `privacy` is used to associate a P3P [P3P 1.0] policy reference with a particular form.

Example: XML Representation: `<privacy>`

```
<privacy
  xlink:href = xsd:anyURI
>
  <!-- Content: (##empty) -->
</privacy>
```

xlink:href - Optional link to an externally defined P3P policyref file (not an actual policy).

3.8 XForms and XLink

XForms uses XLink [XLink] for linking and for defining an explicit relationship between resources that may be either local or remote. To this end, the XForms schema references the XLink namespace with sensible defaults. Other than in the case of attribute `xlink:href`, form authors will not be required to explicitly write XLink-specific elements or attributes.

All XLinks in XForms are simple links. For further details, see **3.8.1 XLink Conformance and Examples**.

3.8.1 XLink Conformance and Examples

An XForms processor is not required to implement full XLink—correct behavior of the `xlink:href` attribute (as defined in this chapter) is sufficient. For example, an XForms Processor must accept and correctly process the schema in both of the following:

Example: External schema constraints

```
<xforms:model>
  <xforms:schema xlink:href="URI-to-remote-schema.xsd" />
</xforms:model>
```

Example: Inline schema constraints

```
<xforms:model>
  <xforms:schema>
    <xsd:schema ...>
      <!-- Content: ... -->
    </xsd:schema>
  </xforms:schema>
</xforms:model>
```

This second example is unusual in that the `xforms:schema` element defaults an attribute `xlink:type="simple"` but lacks an `xlink:href` attribute to make the link meaningful. In this situation, the XForms Processor should switch from `simple` mode to `none` mode for the element lacking attribute `xlink:href`. For compatibility with XLink, the second example should be explicitly authored as follows:

Example: Inline schema constraints, with explicit `xlink:type`

```
<xforms:model>
  <xforms:schema xlink:type="none">
    <xsd:schema...>
      <!-- Content: ... -->
    </xsd:schema>
  </xforms:schema>
</xforms:model>
```

Notice the explicit override of the `xlink:type` attribute.

If both inline content and external reference is provided, a processor must use the external reference and ignore the inline content.

4 Processing Model

This chapter defines the XForms processing model declaratively by enumerating the various states attained by an XForms processor and the possible state transitions that exist in each of these states. The chapter enumerates the pre-conditions and post-conditions that must be satisfied in each of these states. XForms Processors may be implemented in any manner, so long as the end results are identical to that described in this chapter.

The XForms processing model consists of the following three phases:

- initialization
- User interaction
- Submission

Each of these phases is further subdivided as explained in detail in subsequent sections of this chapter. State transitions in the processing model occur when specific events are received, and the event handler that processes the event determines the new state.

4.1 Events Overview

XForms processing is defined in terms of events, event handlers, and event responses. XForms uses the events system defined in [DOM2 Events], with a event Capture phase, arrival of the event at its Target, and finally the event Bubbling Phase.

Events editorial whiteboard:

If the submit event targets submitInfo (which seems reasonable), we should allow XForms Actions as children of <submitInfo>. (Accepted)

Did we get rid of destruct? I couldn't find a reference either way. (No decision)

4.2 Initialization Events

This section defines the various stages of the *initialization* phase. The processing application begins initialization by dispatching an event `xforms:modelConstruct` to each XForms Model in the containing document.

4.2.1 xforms:modelConstruct

Dispatched in response to: XForms Processor initialization.

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

Default processing for this event results in the following:

1. Schema is loaded, if any.
2. An XPath data model is constructed from the instance data, according to the following rules:
   1. From an external source
   2. If there is no reference to an external instance, from an inline instance

   Note:

   If neither of these are supplied, the instance is constructed from the user interface, during user interface construction.
3. Following this, an `xforms:modelInitialize` event is dispatched to element `model`.

4.2.2 xforms:modelInitialize

Dispatched in response to: completion of `xforms:modelConstruct` processing.

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

Default processing for this event results in the following:

1. The instance data has been structurally validated against the Schema, if any. If structural validation fails, all XForms processing for this containing document halts.
2. If applicable, P3P has been initialized. [P3P 1.0]
3. The instance data has been constructed.
4. The `xforms:initializeDone` event is dispatched to the `model` element after initialization of that model element is completed but before rendering of the UI has started.
5. After all XForms Models are initialized, the host must dispatch an `xforms:UIInitialize` event to each `model` element.

4.2.3 xforms:initializeDone

Dispatched in response to: `xforms:modelInitialize` processing.

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

Default processing for this event results in the following:

4.2.4 xforms:UIInitialize

Dispatched in response to: XForms Processor user interface initialization.

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

Default processing for this event results in the following:

The host processor traverses the containing document, and for each form control, dispatches a xforms:formControlInitialize event to the form control.

4.2.5 xforms:formControlInitialize

Dispatched in response to: xforms:UIInitialize processing.

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

Default processing for this event results in the following:

If the referenced model doesn't have instance data, it is created by following the rules for default instance data described below. In the absence of a model, instance data items are treated as having type `xsd:string`. As each user interface control is processed, an instance data element node is created by using the binding expression from the user interface control as the `name`. The resulting instance data may be multiply rooted, and is intended only as a representation of a sequence of name-value pairs.

4.3 Interaction Events

4.3.1 DOM Mutation Events

Dispatched in response to: any change in the instance data.

Target: instance data node

Bubbles: Yes

Cancelable: No

Context Info: varies

In implementations that support the DOM, standard DOM mutation events should be dispatched to the changing target nodes whenever the instance data changes. Note that script, using the method getInstanceDocument() and a tree-walk, is required to associate event handlers with the instance data.

4.3.2 xforms:next and xforms:previous

Dispatched in response to: user request to navigate to the next or previous form control.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for these events results in the following: Navigation according to the default navigation order. For example, on a keyboard interface, "tab" would typically generate an `xforms:next` event, while "shift+tab" would generate an `xforms:previous` event.

Navigation is determined on a containing document-wide basis. The basic unit of navigation is the form control. The <`group`>, <`repeat`>, and <`switch`> structures also serve as navigation units, but instead of providing a single navigation point, they create a local navigation context for child form controls (and possibly other substructures). The navigation sequence is determined as follows:

1. Those navigation units that support `navIndex` and assign a positive value to it are navigated first.
    1. Outermost navigation units are navigated in increasing order of the `navIndex` value. Values need not be sequential nor must they begin with any particular value. Navigation units that have identical `navIndex` values are be navigated in document order.
    2. Ancestor navigation units establish a local navigation sequence. All navigation units within a local sequence are navigated, in increasing order of the `navIndex` value, before any outside the local sequence are navigated. Navigation units that have identical `navIndex` values are navigated in document order.
2. Those form controls that do not supply `navIndex` or supply a value of "0" are navigated next. These form controls are navigated in document order.
3. Those form controls that are disabled, hidden, or not `relevant` are assigned a relative order in the overall sequence but do not participate as navigable controls.
4. The navigation sequence past the last form control (or before the first) is undefined. XForms Processors may cycle back to the first/last control, remove focus from the form, or other possibilities.

4.3.3 xforms:focus and xforms:blur

Dispatched in response to: a form control gaining or losing focus through any means.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for these events results in the following: None; notification events only.

4.3.4 xforms:activate

Dispatched in response to: the "default action request" for a form control, for instance pressing a button or hitting enter.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following: None; notification event only.

4.3.5 xforms:valueChanging

Dispatched in response to: an interactive change to an instance data node bound to a form control.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Certain form controls allow interactive response without finalizing on a value. Examples of this include edit boxes (users can type various characters before navigating away) and slider controls (users can be continuously adjusting the value before releasing at a certain value). Interactive temporary values such as this are expressly allowed to be "invalid", that is outside the permissible value space. This is because incomplete data may be present while the user is entering transitional values.

Example: A partially entered credit card value of "3" is not valid because it doesn't (yet) have enough characters. This is permitted temporarily, as long as the user remains on the form control. XForms Full Processors would update/refresh on every character. XForms Basic Processors would typically only update/refresh on the final value.

Default processing for this event results in the following:

1. If the partial value meets all validity constraints, it is reflected in the instance data. If not, the instance data remains as it was before processing this event.
2. Event `recalculate` has been dispatched to element `model`.
3. Event `refresh` has been dispatched to element `model`.

Implementations that choose to implement `valueChanging` are expected optimize processing (for instance not flashing the entire screen for each character entered, etc.).

Note:

XForms Basic processors are not required to generate or respond to these events.

4.3.6 xforms:valueChanged

Dispatched in response to: a change to an instance data node bound to a form control, when the user navigates away from the form control.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following:

1. The value from the form control is reflected in the instance data.
2. Event `revalidate` has been dispatched to element `model`.
3. Event `recalculate` has been dispatched to element `model`.
4. Event `refresh` has been dispatched to element `model`.

4.3.7 xforms:scrollFirst

Dispatched in response to: a repeat view is scrolled past the beginning of the repeat items.

Target: `repeat`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following: None; notification event only.

4.3.8 xforms:scrollLast

Dispatched in response to: a repeat view is scrolled past the end of the repeat items.

Target: `repeat`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following: None; notification event only.

4.3.9 xforms:insert and xforms:delete

Dispatched in response to: A event handler invoking an XForms Action `insert` or `delete`.

Target: `instance`

Bubbles: Yes

Cancelable: Yes

Context Info: Path expression used for insert/delete.

Default processing for these events results in the following: None; notification event only.

4.3.10 xforms:select and xforms:deselect

Dispatched in response to: an item in a `selectOne`, `selectMany`, or `switch` becoming selected or deselected.

Target: form control or `switch`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following: None; notification event only.

4.3.11 xforms:help and xforms:hint

Dispatched in response to: a user request for help or hint information.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for these events results in the following: None; notification event only. User agents may provide default help or hint messages.

4.3.12 xforms:alert

Dispatched in response to: a form control failing validation.

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following: An error message displayed, informing the user of the action needed to make the form control valid.

4.3.13 xforms:valid

Dispatched in response to: a form control becoming valid with respect to the bound instance data.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

Default processing for this event results in the following: None; notification event only.

4.3.14 xforms:invalid

Dispatched in response to: a form control becoming invalid with respect to the bound instance data.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

Default processing for this event results in the following:

- Event `alert` is dispatched to the form control.

4.3.15 xforms:refresh

Dispatched in response to: a request to update all form controls associated with a particular XForms Model.

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following: The user interface will reflect the state of the model. This means:

- All form controls show the current value corresponding to the bound instance data.
- All form controls show the validity state of the corresponding bound instance data.
- Any form control associated with a model item property `relevant` evaluating to `false` is disabled/hidden/etc.

4.3.16 xforms:revalidate

Dispatched in response to: a request to revalidate one or all form controls associated with a particular XForms Model.

Target: `model` or a form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following:

Revalidation may occur targeted to a context form control. The default handling for this event must satisfy the following conditions:

1. The bound instance data node is checked against any bound Schema Constraints. If any fail, the context form control is considered invalid.
2. The bound instance data node is checked against any bound XForms Constraints. If any fail, the context form control is considered invalid.
3. If the context form control is invalid, the XForms Processor must dispatch event `invalid` to the context form control. Otherwise, event `valid` must be dispatched to the form control.

When element `model` is targeted by this event, the above is applied to every form control in document order.

4.3.17 xforms:recalculate

Dispatched in response to: a request to recalculate all calculations associated with a particular XForms Model.

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following:

An XPath expression is bound either to the value or to a model item property (e.g., `required`, `relevant`) of one or more instance nodes. The combination of an XPath expression with a single instance node's value or model item property is considered as a single computational unit, a **compute**, for the purposes of recalculation.

When it is time to recalculate a compute, the XPath expression is evaluated in the context of the instance node whose value or model item property is associated with the compute. The XPath expression may **reference** or **refer to** another instance node, in which case the value of the instance node is **referenced**. Each referenced instance node has as **dependents** those computes which directly refer to the instance node. Self-references are explicitly ignored, i.e., if an expression associated with a compute refers to the instance node associated with the compute, then the instance node does not take itself as a dependent. A compute is **computationally dependent** on an instance node (whose value may or may not be computed) if there is a path of dependents leading from the instance node through zero or more other instance nodes to the compute. A compute is part of a **circular dependency** if it is computationally dependent on itself.

When a recalculation event begins, there will be a list *L* of one or more instance nodes whose values have been changed, e.g., by user input being propagated to the instance.

1. An XForms processor must not recalculate computes that are not computationally dependent on one or more of the elements in *L*.
2. An XForms processor must perform a single recalculation of each compute that is computationally dependent on one or more of the elements in *L*.
3. An XForms processor must recalculate a compute *C* after recalculating all computes of instance nodes on which *C* is computationally dependent. (Equivalently, an XForms processor must recalculate a compute *C* before recalculating any compute that is computationally dependent on the instance node associated with *C*.)
4. Finally, if a compute is part of a circular dependency and also computationally dependent on an element in *L*, then an XForms processor MUST report an exception.

C Recalculation Sequence Algorithm describes one possible method for achieving the desired recalculation behavior.

4.3.18 xforms:reset

Dispatched in response to: a user request to reset the instance data.

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following:

1. All of the instance data is selected for resetting.

2. New instance data for the selected instance data is prepared, based on the `instance` element associated with the current `model` element, according to the rules for initialization above.
3. The selected instance data is replaced with the new instance data.

4.4 XForms Submit

Form filling experience ends with submitting the form, or perhaps starting over. The XForms processing for these events are covered here. The following sections describe how the instance data is prepared for submission.

4.4.1 xforms:submit

Dispatched in response to: a user request to submit the instance data.

Target: `submitInfo`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Default processing for this event results in the following:

1. A node from the instance data is selected, based on the attribute `ref` on element `submitInfo`. This node and all child nodes, are considered for the remainder of the submit process.
2. All selected instance data is revalidated. Any invalid instance data stops submit processing.
3. Selected instance data is serialized according to one of the processes defined below, as indicated by element `submitInfo` attributes `mediaType` and `mediaTypeExtension`. Nodes that have an associated relevant constraints that evaluates to false are not serialized.
4. Instance data is delivered over the network using the network protocol indicated by element `submitInfo` attribute `method`.
   Note:

   The HTTP "get" protocol is deprecated for use in form submission. Form authors should use "post" for greater compatibility.

5. The response returned from the submission is applied as follows: if element `submitInfo` attribute `replace` has the value of "`all`", the entire containing document is replaced. If the attribute value is "`instance`", the response is parsed as XML and the internal instance data is replaced with the result, using the same processing as remote instance data retrieved through `xlink:href`, and the `xforms:initialize` event is dispatched to element `model`. Behaviors of other possible values for attribute `replace` are not defined in this specification.

   Under no circumstances may more than a single concurrent submit process be under way for a particular XForms Model.

4.4.2 application/x-www-form-urlencoded

This format is selected by the string `application/x-www-form-urlencoded` in element `submitInfo` attribute `mediaType`.

Note:

This serialization format is deprecated, and will be removed in a future version of the XForms specification. For greater compatibility with XML and non-western characters, form authors should choose a different serialization format.

This format is intended to facilitate the integration of XForms into HTML forms processing environments, and represents an extension of the [XHTML 1.0] form content type of the same name with extensions to expresses the hierarchical nature of instance data.

This format is not suitable for the persistence of binary content. Therefore, it is recommended that XForms capable of containing binary content use either the multipart/form-data (**4.4.3 multipart/form-data**) or text/xml (**4.4.4 text/xml**) formats.

Issue (issue-urlencoding-mods):
Modifications to urlencoding process

The urlencoding technique given here does not exactly match how legacy implementations produce urlencoded data. (In particular, we are adding contextual information with slashes and multiple location-steps) Will this approach interfere with legacy implementations?

Issue (issue-utf8-encoding):

Under discussion is the intent to have the data be UTF8 encoded; however, this is dependent upon IETF developments. Would UTF8 meet the needs of the forms community?

Instance data is urlencoded with the following rules:

1. Each element node is visited in document order. If the element contains only a single node, it is selected for inclusion. Note that attribute information is not preserved.
2. Elements selected for inclusion are encoded as "EltName=value;", where "=" and ";" are literal characters, "EltName" represents the element local name, and "value" represents the contents of the text node. Note that contextual path information is not preserved, nor are namespace prefixes, and multiple elements might have the same name.
3. All such encodings are concatenated, maintaining document order. The resulting string is urlencoded, as in HTML processing.

Example:

Example: application/x-www-form-urlencoded

```
FirstName=Roland;
```

This format consists of simple name-value pairs.

```
<PersonName title="Mr">
  <FirstName>Roland</FirstName>
</PersonName>
```

Here is the instance data for the above example. Note that very little of the data is preserved. Authors desiring greater data integrity should select a different serialization format.

4.4.3 multipart/form-data

This format is selected by the string `multipart/form-data` in element `submitInfo` attribute `mediaType`.

This format is intended to facilitate the integration of XForms into HTML forms processing environments, and represents an extension of the [XHTML 1.0] form content type of the same name that expresses the hierarchical nature of instance data. Unlike the application/x-www-form-urlencoded (**4.4.2 application/x-www-form-urlencoded**) format, this format is suitable for the persistence of binary content.

This format follows the rules of all multipart MIME data streams for form data as outlined in [RFC 2388], with the "name" of each part being the canonical binding expression that references the selected instance data node.

Example:

Example: multipart/form-data

```
Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x
  Content-Disposition: form-data; name="/PersonName/@title"

Mr
--AaB03x
  Content-Disposition: form-data; name="/PersonName/FirstName"

Roland
--AaB03x

...Possibly more data...

--AaB03x-
```

This format consists of sets of a canonical binding expression paired with a value.

```
<PersonName title="Mr">
  <FirstName>Roland</FirstName>
</PersonName>
```

Here is the instance data for the above example.

4.4.4 text/xml

This format is selected by the string `text/xml` in element `submitInfo` attribute `mediaType`.

This format permits the expression of the instance data as XML that is straightforward to process with off-the-shelf XML processing tools. In addition, this format is suitable for the persistence of binary content.

The steps for building this persistence format is as follows:

1. An XML document is produced following the rules of the XML output method defined in XPath [XSLT] section 16 and 16.1, using the values supplied as attributes of the `submitInfo` element.
2. If the selected content of the instance data corresponds to a multiply-rooted data structure (such as a general parsed entity), an the above serialization takes place, after which the serialized instance data is inserted as child elements of the unqualified element `instanceData`.

4.4.4.1 Binary Content

Instance data nodes with values of the types xsd:base64Binary and xsd:hexBinary are specifically allowed, and are included in the serialized data according to the rules defined in [XML Schema part 2]

Issue (issue-instance-metadata):

Where a value within the instance data represents binary content, can we store meta-information with an `xform:mediaType` attribute reflecting the appropriate content type (e.g., "image/jpg")?

4.5 Error Indications

4.5.1 xforms:schemaConstraintsError

Dispatched in response to: instance data becoming schema-invalid.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

Default processing for this event results in the following: None; notification event only. Default error handling may be used.

4.5.2 xforms:traversalError

Dispatched in response to: a failure in link traversal of an xlink:href attribute value.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: The URI that failed to load.

Default processing for this event results in the following: None; notification event only. Default error handling may be used.

4.5.3 xforms:invalidDatatypeError

Dispatched in response to: an invalid parameter passed to an XForms function.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

Default processing for this event results in the following: None; notification event only. Default error handling may be used.

5 Datatypes

This chapter defines the datatypes used in defining an XForms model.

5.1 XML Schema Built-in Datatypes

XForms includes all XML Schema datatypes. Concepts value space, lexical space and constraining facets are as specified in [XML Schema part 2]. XML Schema features used in XForms are divided into two modules, called *Basic* and *Full*. Base types included in module *basic* are marked with an asterisk *. Datatypes *derived by restriction* and *derived by list* from these base types are also included in the basic module.

Built-in primitive types:

duration *
dateTime *
time *
date *
gYearMonth *
gYear *
gMonthDay *
gDay *
gMonth *
string *
boolean *
base64Binary *
hexBinary
float
decimal *
double
anyURI *
QName
NOTATION

Built-in derived types:

normalizedString
token
language
Name
NCName
ID
IDREF
IDREFS
ENTITY
ENTITIES
NMTOKEN
NMTOKENS
integer *
nonPositiveInteger *
negativeInteger *
long *
int *
short *
byte *
nonNegativeInteger *
unsignedLong *
unsignedInt *
unsignedShort *
unsignedByte *
positiveInteger *

5.2 XForms Datatypes

The Schema for XForms derives the following types to facilitate defining `model` in XForms. These types are included in XForms Basic.

5.2.1 xforms:listItem

This datatype serves as a base for the listItem datatype. The value space for listItem permits one or more characters valid for xsd:string, except whitespace characters.

5.2.2 xforms:listItems

XForms includes form controls that produce simpleType list content. This is facilitated by defining a `derived-by-list` datatype. The value space for listItems is defined by list-derivation from listItem.

Note:

In most cases, it is better to use markup to distinguish items in a list. See **8.11.3 itemset**.

6 Constraints

This chapter defines constraints that can be bound to form data. The combination of these constraints with an instance data node is called a model item. Taken together, these constraints are called model item constraints. The term Schema constraint refers only to XML Schema datatype constraints, while the term XForms constraint refers to XForms-specific constraints defined in the following section.

6.1 XForms Constraints

XForms constraints are defined via attributes of element `bind`. There are two kinds of constraints in XForms 1.0 as defined below.

- Fixed constraints are static values that the XForms Processor evaluates only once. Such constraints typically encode type information.
- Computed expressions are XPath expressions that provide a value to the XForms Processor. Such values are recomputed at certain times as specified by the XForms Processing Model (see **4 Processing Model**). These expressions encode dynamic constraints such as the dependency among various data items. Computed expressions are not restricted to examining the value of the instance data node to which they apply. XPath expressions provide the means to traverse the instance data; more complex computations may be encoded as call-outs to external scripts.

The following constraints are available for all model items. For each constraint, the following information is provided:

Description
Computed Expression (yes or no)
Applies to children (inherited by instance data child elements and attributes)
Legal Values
Default Value

6.1.1 type

Description: associates a Schema datatype.

Computed Expression: No.

Applies to children: No.

Legal Values: Any `xsd:QName` representing an in-scope datatype.

Default Value: `xsd:string`.

The effect of this constraint is the same as placing attribute `xsi:type` on the instance data.

6.1.2 readOnly

Description: describes whether the value is restricted from changing. The ability of form controls to have focus and appear in the navigation order is unaffected by this constraint.

Computed Expression: Yes.

Applies to children: Yes.

Legal Values: Any expression that is convertible to `boolean`.

Default Value: `false`.

When evaluating to `true`, this constraint indicates that the XForms Processor should not allow any changes to the bound instance data node.

In addition to restricting value changes, the `readOnly` constraint provides a hint to the XForms User Interface. Form controls bound to instance data with the `readOnly` constraint should indicate that entering or changing the value is not allowed. This specification does not define any effect on visibility, focus, or navigation order.

6.1.3 required

Description: describes whether a value is required before the instance data is submitted.

Computed Expression: Yes.

Applies to children: Yes.

Legal Values: Any expression that is convertible to `boolean`.

Default Value: `false`.

A form may *require* certain values, and this *requirement* may be dynamic. When evaluating to `true`, this constraint indicates that a non-empty instance data node is required before a submission of instance data can occur. Non-empty is defined as:

1. If the bound instance data node is an element, the element must not have the `xsi:nil` attribute set to `true`.
2. The value of the bound instance data node must be convertible to an XPath `string` with a length greater than zero.

Except as noted below, the `required` constraint does not provide a hint to the XForms User Interface regarding visibility, focus, or navigation order. XForms authors are strongly encouraged to make sure that form controls that accept `required` data are visible. An XForms Processor may provide an indication that a form control is required, and may provide immediate feedback, including limiting navigation. Chapter **4 Processing Model** contains details on how the XForms Processor enforces required values.

6.1.4 relevant

Description: indicates whether the model item is currently *relevant*. Instance data nodes with `relevant=false` are not serialized for submission.

Computed Expression: Yes.

Applies to children: Yes.

Legal Values: Any expression that is convertible to `boolean`.

Default Value: `true`.

Many forms have data entry fields that depend on other conditions. For example, a form might ask whether the respondent owns a car. It is only appropriate to ask for further information about their car if they have indicated that they own one.

Constraint `relevant` provides hints to the XForms User Interface regarding visibility, focus, and navigation order. In general, when `true`, associated form controls should be made visible. When `false`, associated form controls should be made unavailable, removed from the navigation order, and not allowed focus.

The following table shows the user interface interaction between `required` and `relevant`.

6.1.5 calculate

Description: supplies an expression used to calculate the value of the associated instance data node.

Computed Expression: Yes.

Applies to children: No.

Legal Values: Any XPath expression

Default Value: none.

An XForms Model may include model items that are computed from other values. For example, the sum over line items for quantity times unit price, or the amount of tax to be paid on an order. Such computed value can be expressed as a computed expression using the values of other model items. The XForms Processing Model indicates when and how the calculation is performed.

6.1.6 isValid

Description: specifies a predicate that needs to be satisfied for the associated instance data node to be considered valid.

Computed Expression: Yes.

Applies to children: No.

Legal Values: Any expression that is convertible to `boolean`.

Default Value: `true`.

When evaluating to `false`, the associated model item is not valid; the converse is not necessarily true. Chapter **4 Processing Model** describes details such as immediate validation versus validation upon submit.

The XForms User Interface may indicate the validity of a form control.

6.1.7 maxOccurs

Description: for repeating structures, indicates the maximum number of allowed child elements.

Computed Expression: No.

Applies to children: No.

Legal Values: `xsd:integer` or `"unbounded"`.

Default Value: `"unbounded"`.

For model item elements that are repeated, this optional constraint specifies a maximum number of allowed child elements.

6.1.8 minOccurs

Description: for repeating structures, indicates the minimum number of allowed child elements.

Computed Expression: No.

Applies to children: No.

Legal Values: `xsd:integer`.

Default Value: 0.

For model item elements that are repeated, this optional constraint specifies a minimum number of allowed child elements.

6.2 Schema Constraints

Chapter **5 Datatypes** described how XForms uses the XML Schema datatype system to constrain the value space of data values collected by an XForm. Such datatype constraints can be provided via an XML Schema. Alternatively, this section lists various mechanisms for attaching type constraints to instance data. Attributes `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` are ignored for purposes for locating a Schema. XForms Basic processors have restricted Schema processing requirements as defined in **11.1.1 XForms Basic**.

6.2.1 Atomic Datatype

The XForms Processing Model applies XML Schema facets as part of the validation process. At the simplest level, it is necessary to associate a set of facets (through a Schema datatype) with a model item. This has the effect of restricting the allowable values of the associated instance data node to valid representations of the lexical space of the datatype.

The set of facets may be associated with a model item in one of the following ways (only the first that applies is used, and if multiple type constraints apply to the same node, the first definition in document order is used).

1. An XML Schema associated with the instance data.
2. An XML Schema `xsi:type` attribute in the instance data.
3. An XForms `type` constraint associated with the instance data node using XForms binding.
4. If no type constraint is provided, the data instance node defaults to `type=xsd:string` (default to string rule).

The following declares a datatype based on `xsd:string` with an additional constraining facet.

Example: Type Constraint Using Schema.

```
<xsd:simpleType name="nonEmptyString">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
  </xsd:restriction>
</xsd:simpleType>
```

This new datatype would then be associated with one or more model items through one of the methods outlined here.

Example: Attaching A Type Constraint

```
<my:first-name xsi:type="nonEmptyString"/>
```

This defines element `first-name` to be of type `nonEmptyString`.

Example: Attaching Type Constraint Using XForms Binding

```
<instance>
  <my:first-name />
</instance>
<bind type="nonEmptyString" ref="/my:first-name"/>
```

Here, we have attached type information to element `first-name` via element `bind`. This enables the XForms author extend external Schemas that she does not have the ability to change.

6.3 Additional Schema Examples

The following non-normative sections illustrate mapping between Schema concepts and data structures commonly used in form authoring.

6.3.1 Closed Enumeration

It is often necessary to restrict the allowable values of the associated instance data node to a closed list of alternatives, e.g., when asking for a credit card type. Here is a schema fragment that declares a datatype that allows enumerated values of an `xsd:string`.

Example: Closed Enumeration

```
<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="MusterCard"/>
    <xsd:enumeration value="Donor'sClub"/>
    <xsd:enumeration value="WildExpress"/>
    <xsd:enumeration value="EntryPermit"/>
  </xsd:restriction>
</xsd:simpleType>
```

6.3.2 Open Enumeration

A special case of enumerated datatypes is the common form design pattern of a list with an 'other, please specify' choice. This is referred to as an open enumeration.

Declaring an open enumeration is possible through a combination of union and enumeration.

Example: Open Enumeration

```
<xsd:simpleType>
  <xsd:union memberTypes="xsd:string">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="MusterCard"/>
        <xsd:enumeration value="Donor'sClub"/>
        <xsd:enumeration value="WildExpress"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

6.3.3 Union

It may be desirable to allow an instance data item to be a valid lexical value of one among several datatypes. Unions are defined in XML Schema.

The following defines a datatype that accepts either a `creditCardType` or `bonusProgramType`.

Example: Union Of Types

```
<xsd:simpleType>
  <xsd:union memberTypes="creditCardType bonusProgramType"/>
</xsd:simpleType>
```

6.3.4 Lists

Form controls such as `selectMany` collect more than one value. This corresponds to Schema list datatypes.

The following declares a list-derived datatype.

Example: List Datatype

```
<xsd:simpleType name="listOfMyIntType">
  <xsd:list itemType="xsd:int"/>
</xsd:simpleType>
```

6.4 Binding

Binding is the glue that connects the separate pieces of XForms—here, we use XForms binding to associate instance data with model item constraints.

Binding is specified via binding expressions, which select nodes from the instance data. Binding expressions are based on XPath and are defined in chapter **7 XPath Expressions in XForms**. This section describes how binding expressions are used when defining the XForms model.

6.4.1 bind

Element `bind` operates on a node-set selected from the instance data. Attributes on element `bind` encode XForms constraints to be applied to each node in the node-set.

Example: XML Representation: `<bind>`

```
<bind ref = binding-expression
  <!-- model item constraints -->
  type = xsd:QName
  readOnly = model-item-constraint
  required = model-item-constraint
  relevant = model-item-constraint
  isValid = model-item-constraint
  calculate = model-item-constraint
  maxOccurs = xsd:nonNegativeInteger or 'unbounded'
  minOccurs = xsd:nonNegativeInteger
>
  <!-- Content: (bind)* -->
</bind>
```

ref - A binding expression that selects the set of nodes that this `bind` operates on.
model item constraints Model item constraints as defined in **6.1 XForms Constraints**.

Each bind element selects a node-set from the instance data, and applies the specified constraints. When additional nodes are added through action `insert`, the newly added nodes are included in any node-sets matched by binding expressions—see action `insert` in **10.11 insert**.

6.4.2 Rules For Binding Expressions

Not every possible XPath expression is acceptable as a binding expression. The following rules are used to limit the range of XPath expressions that can appear as valid binding expressions.

1. **No dynamic predicates.** Predicates are permitted, but such predicates must not depend on other form settings. Here are a few examples to illustrate this.
   Example: Permissible Binding Expressions

   ```
   permitted: elem
     permitted: elem[1]
     permitted: elem[last()]
     permitted: elem[@id="zip"]  if @id is not bound to a form control
     forbidden: elem[@attr="xy"]  if @attr is bound to a form control
   ```

2. **No invocation of any function that returns a node-set.** Function calls are permitted, but not any that return a node-set.
3. **No invocation of any function with side-effects.** All functions defined in the XForms specification are side-effect-free. Any extension functions should also be side-effect-free.

Upon detecting a binding expression that violates any of the above constraints, form processing terminates with a fatal error.

6.4.3 Binding References

Binding references can be used to bind form controls to the underlying data instance as described in **8.12.2 Single Node Binding Attributes** and **8.12.3 Nodeset Binding Attributes**. Different attribute names, `ref` and `nodeset` distinguish between a single node and a node-set respectively.

First node rule: When a single-node binding expression selects a node-set of size > 1, the first node in the node-set is used. This has no effect on the individual nodes nor the set of nodes selected by any particular `bind` element.

Consider a document with the following XForms declarations:

Example: First Node Rule

```
<xforms:model id="orders">
  <xforms:instance xmlns="">
    <orderForm>
      <shipTo>
        <firstName>John</firstName>
      </shipTo>
    </orderForm>
  </xforms:instance>
  <xforms:bind ref="/orderForm/shipTo/firstName" id="fn" />
</xforms:model>
```

The following examples show three ways of binding user interface control `xforms:input` to instance element `firstName` declared in the model shown above.

Example: UI Binding Using Attribute `ref`

```
<xforms:input ref="/orderForm/shipTo/firstName">...
```

Example: UI Binding Using Attribute `bind`

```
<xforms:input bind="fn">...
```

Example: Specifies Model Containing The Instance Explicitly

```
<xforms:input model="orders" ref="/orderForm/shipTo/firstName">...
```

The XForms binding mechanism allows other XML vocabularies to bind user interface controls to an XForms model using any of the techniques shown here. As an example, XForms binding attribute `bind` might be used within legacy HTML user interface controls as shown below.

Example: XForms Binding In Legacy HTML User Interface Controls

```
<html:input type="text" name="..." xforms:bind="fn"/>
```

7 XPath Expressions in XForms

XForms uses XPath to address instance data nodes in binding expressions, to express constraints, and to specify calculations.

7.1 XPath Datatypes

XPath data types are used only in Binding expressions and computed expressions. XForms uses XPath datatypes `boolean`, `string`, `number` and `node-set`. A future version of XForms is expected to use XPath 2.0, which includes support for XML Schema datatypes.

7.2 Instance Data

For each `model` element, the XForms processor maintains the state in an internal structure called instance data that conforms to the XPath Data Model [XPath 1.0]. Elements and attributes in the instance data may have namespace information associated with them, as defined in the XPath Data Model. Unless otherwise specified, all instance data elements and attributes are unqualified. In addition, XForms processors must provide DOM access to this instance data via the interface defined below.

interface XFormsModelElement : org.w3c.dom.Element

The method getInstanceDocument returns a DOM Document that corresponds to the instance data associated with this XForms Model.

Return value: org.w3c.dom.Document

raises (DOMException); if there is no model with the specified model-id.

If the instance data is multiply rooted, the returned document has unqualified element `instanceData` as the `documentElement`, with the content of the XForms Model as children.

7.3 Evaluation Context

Within XForms, XPath expressions reference abstract instance data (using the "path" portion of XPath), instead of a concrete XML document. This reference is called a binding expression in this specification. Every XPath expression requires an evaluation context. The following rules are used in determining evaluation context when evaluating elements containing binding expressions in XForms:

1. The context node for outermost binding elements is the XPath root (`/`). A " **binding element**" is any element other than `bind` that is explicitly allowed to have a binding expression attribute. A binding element is "**outermost**" when the node-set returned by the XPath expression `ancestor::*` includes no binding element nodes.
2. The context node for non-outermost binding elements is the first node of the binding expression of the immediately enclosing element. An element is "**immediately enclosing**" when it is the first binding element node in the node-set returned by the XPath expression `ancestor::*`. This is also referred to as "scoped resolution".
3. The context node for the `ref` attribute on `bind` is the XPath root. The context node for computed expressions (occurring on element `bind`) is the first node of the node-set returned from the binding expression in the sibling `ref` attribute.
4. The context size and position are both exactly 1.
5. No variable bindings are in place.
6. The available function library is defined below, plus any function names declared in attribute `extensionFunctions` on element `model`.
7. Any namespace declarations in scope for the attribute that defines the expression are applied to the expression.

Example: Binding Expression Context Nodes

```
<group ref="level1/level2/level3">
  <selectOne ref="elem" ... />
  <selectOne ref="@attr" ... />
</group>
```

In this example, the `group` has a binding expression of `level1/level2/level3`. According to the rules above, this outermost element would have a context node of `/`, which is the root of the instance data, or the parent to the `elem` element. Both of the `selectOne`s then inherit a context node from their parent, the context node being `/level1/level2/level3`. Based on this, the `selectOne` binding expressions evaluate respectively to `/level1/level2/level3/elem` and `/level1/level2/level3/@attr`. Matching instance data follows:

Example: Sample Instance

```
<level1>
  <level2>
    <level3 attr="xyz">
      <elem>xyz</elem>
    </level3>
  </level2>
```

`</level1>`

7.4 XForms Core Function Library

The XForms Core Function Library includes the entire [XPath 1.0] Core Function Library, including operations on node-sets, strings, numbers, and booleans.

This section defines a set of required functions for use within XForms.

7.4.1 Boolean Methods

7.4.1.1 boolean-from-string()

boolean **boolean-from-string**(`string`)

Function `boolean-from-string` returns `true` if the required parameter `string` is "true", or `false` if parameter `string` is "false". This is useful when referencing a Schema `xsd:boolean` datatype in an XPath expression. If the parameter string matches neither "true" nor "false", according to a case-insensitive comparison, processing stops with a fatal error.

7.4.1.2 if()

string **if**(`boolean`, `string`, `string`)

Function `if` evaluates the first parameter as boolean, returning the second parameter when `true`, otherwise the third parameter.

7.4.2 Number Methods

Note:

The XPath number datatype and associated methods and operators use IEEE specified representations. XForms Basic Processors are not required to use IEEE, and thus might yield slightly different results.

7.4.2.1 avg()

number **avg**(`node-set`)

Function `avg` returns the arithmetic average of the result of converting the string-values of each node in the argument node-set to a number. The sum is computed with `sum()`, and divided with `div` by the value computed with `count()`.

7.4.2.2 min()

number **min**(`node-set`)

Function `min` returns the minimum value of the result of converting the string-values of each node in argument `node-set` to a number. "Minimum" is determined with the `<` operator. If the parameter is an empty node set, the return value is NaN.

7.4.2.3 max()

number **max**(`node-set`)

Function `max` returns the maximum value of the result of converting the string-values of each node in argument `node-set` to a number. "Maximum" is determined with the `<` operator. If the parameter is an empty node set, the return value is NaN.

7.4.2.4 count-non-empty()

number **count-non-empty**(, `node-set`)

Function `count-non-empty` returns the number of non-empty nodes in argument `node-set`. A node is considered non-empty if it is convertible into a string with a greater-than zero length.

7.4.2.5 cursor()

number **cursor**(, `string`)

Function `cursor` takes a string argument that is the `idref` of a `repeat` and returns the current position of the repeat cursor for the identified `repeat`—see **9.3 repeat** for details on `repeat` and its associated repeat cursor. If the specified argument does not identify a `repeat`, this function throws an error.

Example: cursor

```
<xforms:button>
                  <xforms:caption>Add to Shopping
                  Cart</xforms:caption> <xforms:insert
                  ev:event="ev:activate" position="after"
                  nodeset="items/item"
                  at="cursor('cartUI')"/>
                  </xforms:button>
```

7.4.3 String Methods

7.4.3.1 `property()`

string **property**(, `string`)

Function `property` returns the XForms Property named by the string parameter.

The following properties are available for reading (but not modification).

version

`version` is defined as the string "`1.0`" for XForms 1.0

conformance-level

`conformance-level` strings are defined in **11 Conformance**.

Example: property

```
<xforms:instance>
                  ...  <xforms:bind
                  ref="info/xforms-version"
                  calculate="property('version')"/> ...
                  </xforms:instance>
```

7.4.3.2 now()

string **now**()

The `now` function returns the current system date and time as a string value in the canonical Schema `xsd:dateTime` format. If time zone information is available, it is included (normalized to UTC).

7.4.4 Extension Functions

XForms documents may use additional XPath extension functions beyond those described here. The names of any such extension functions must be declared in attribute `extensionFunctions` on element `model`. Such declarations are used by the XForms processor to check against available extension functions. XForms processors perform this check at the time the document is loaded, and stop processing by signalling a fatal error if the XForms document declares an extension function for which the processor does not have an implementation.

Note:

Explicitly declaring extension functions enables XForms processors detect the use of unimplemented extension functions at document load-time, rather than throwing a fatal error during user interaction. Failure by authors to declare extension functions will result in an XForms processor potentially halting processing during user interaction with a fatal error.

8 Form Controls

XForms User Interface controls—form controls—are declared using markup elements, and their behavior refined via markup attributes. This markup may be decorated with `class` attributes that can be used in CSS stylesheets to deliver a customized look and feel. XForms user interface controls are bound to the underlying instance data using binding attributes as defined in the chapter **6 Constraints**.

Form controls enable accessibility by taking a uniform approach to such features as captions, help text, tabbing and keyboard shortcuts. Internationalization issues are addressed by following the same design principles as in XHTML. All form controls are suitable for styling using Aural CSS (ACSS) style properties.

Form controls encapsulate high-level semantics without sacrificing the ability to deliver real implementations. For instance, form controls `selectOne` and `selectMany` enable the user *select one or more items from a set*. These form controls distinguish the functional aspects of the underlying control from the presentational aspects (through `class` attributes) and behavior (through XForms Action elements). This separation enables the expression of the intent underlying a particular form control—see [AUI97] for a definition of such high-level user interaction primitives.

Form controls when rendered display the underlying data values to which they are bound. While the data presented to the user through a form control must directly correspond to the bound instance data, the display representation is not required to exactly match the lexical value. For example, user agents should apply appropriate conventions to the display of dates, times, durations and numeric values including separator characters.

XForms user interface controls use common attributes and elements that are defined in (**8.12 Common Markup**). Sections in this chapter define the various form controls by specifying the following:

Description
Examples
Data Binding Restrictions
Implementation Requirements
XML Representation

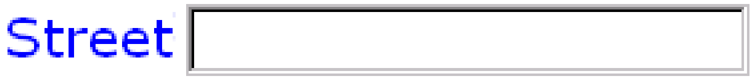
8.1 input

Description: This form control enables free-form data entry.

```
<input ref="order/shipTo/street" class="streetAddress">
  <caption>Street</caption>
  <hint>Please enter the number and street name</hint>
</input>
```

In the above, the `class` attribute can be used by a stylesheet to specify the display size of the form control. Note that the constraints on how much text can be input are obtained from the underlying XForms Model definition and not from these display properties.

A graphical browser might render the above example as follows:



Data Binding Restrictions: Binds to any simpleContent (except `xsd:base64Binary`, `xsd:hexBinary` or any datatype derived from these).

Implementation Requirements: Must allow entry of a lexical value for the bound datatype. Implementations should provide the most convenient means possible for entry of datatypes and take into account localization and internationalization issues such as representation of numbers. For example, an `input` bound to an instance data node of type `Date` might provide a calendar control to enter dates; similarly, an input control bound to data instance of type `boolean` might be rendered as a simple checkbox.

Example: XML Representation: <input>

```
<input
  (single node binding attributes)
  (common attributes)
  inputMode = xsd:string
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</input>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**

common attributes defined in **8.12.1 Common Attributes**

inputMode - this form control accepts an input mode hint. **D Input Modes**

Note:

Notice that not binding any user interface to a piece of instance data results in an *hidden* form control in XForms; consequently, there is no need to explicitly define input form controls with `type="hidden"` as in HTML.

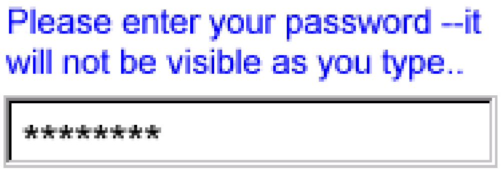
8.2 secret

Description: This form control is used for entering information that is considered sensitive, and thus not echoed to a visual or aural display as it is being entered, e.g., password entry.

Example: Password Entry

```
<secret ref="/login/password">
  <caption>Password</caption>
  <hint>Please enter your password --it will not
    be visible as you type.</hint>
</secret>
```

A graphical browser might render this form control as follows:


Please enter your password --it will not be visible as you type..

****

Data Binding Restrictions: Identical to `input`.

Implementation Requirements: In general, implementations, including accessibility aids, must render a "*" or similar character instead of the actual characters entered, and thus must not render the entered value of this form control. Note that this provides only a casual level of security; truly sensitive information will require additional security measures outside the scope of XForms.

Example: XML Representation `<secret>`

```
<secret
  (single node binding attributes)
  (common attributes)
  inputMode = xsd:string
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</secret>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**

common attributes defined in **8.12.1 Common Attributes**

inputMode - this form control accepts an input mode hint. **D Input Modes**

8.3 textarea

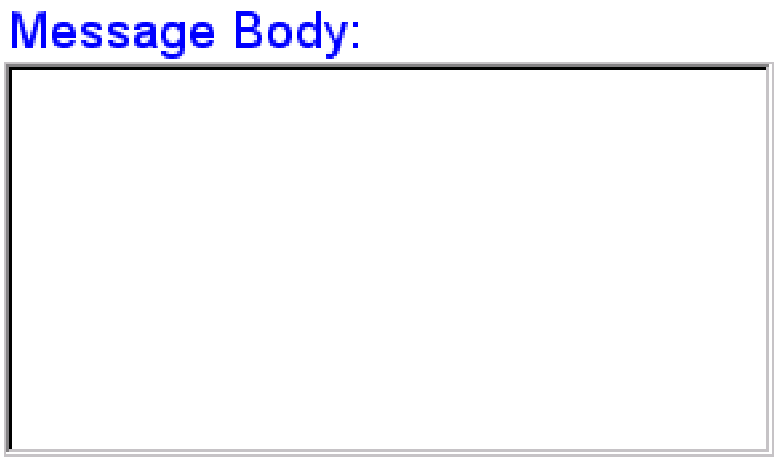
Description: This form control enables free-form data entry and is intended for use in entering multiline content, e.g., the body of an email message.

Example: Email Message Body

```
<textarea ref="message/body" class="messageBody">
  <caption>Message Body</caption>
  <hint>Enter the text of your message here</hint>
</textarea>
```

In the above, the `class` attribute can be used by a stylesheet to specify the display size of the form control. Note that the constraints on how much text can be input are obtained from the underlying XForms Model definition and not from these display properties.

A graphical browser might render the above example as follows:



Data Binding Restrictions: Binds to `xsd:string` or any derived simpleContent.

Implementation Requirements: Must allow entry of a lexical value for the bound datatype, including multiple lines of text.

Example: XML Representation: `<textarea>`

```
<textarea
  (single node binding attributes)
  (common attributes)
  inputMode = xsd:string
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</textarea>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
common attributes defined in **8.12.1 Common Attributes**
inputMode - this form control accepts an input mode hint. **D Input Modes**

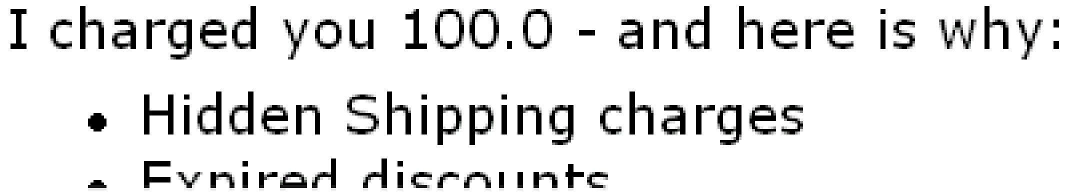
8.4 output

Description: This form control renders a value from the instance data, but provides no means for entering or changing data. It is typically used to display values from the instance, and is treated as `display:inline` for purposes of layout.

Example: Explanatory Message

```
I charged you -
<output ref="order/totalPrice"/>
and here is why:
```

A graphical browser might render an output form control as follows:



I charged you 100.0 - and here is why:

- Hidden Shipping charges
- Expired discounts

Data Binding Restrictions: Binds to any simpleContent.

Implementation Requirements: Must allow display of a lexical value for the bound datatype. Implementations should provide the most convenient means possible for display of datatypes and take into account localization and internationalization issues such as representation of numbers.

Example: XML Representation: `<output>`

```
<output
  (single node binding attributes)
>
  <!-- empty content -->
</output>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**

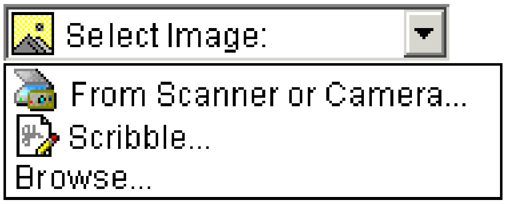
8.5 upload

Description: This form control enables the common feature found on Web sites to upload a file from the local file system, as well as accepting input from various devices including microphones, pens, and digital cameras.

Example: Uploading An Image

```
<upload ref="mail/attach1" mediaType="image/*">
  <caption>Select image:</caption>
</upload>
```

A graphical browser might render this form control as follows:



Data Binding Restrictions: This form control can only be bound to datatypes `xsd:base64Binary` or `xsd:hexBinary`, or types derived by restriction from these.

Implementation Requirements: For suitable mediaTypes:

- Implementations with a file system should support *file upload*—selecting a specific file. The types of files presented by default must reflect the mediaType specified in the XForms Model, for example defaulting to only audio file types in the file dialog when the mediaType is "audio/*". In XForms 1.0, there is a 1:1 binding between a upload form control and one of the `binary` datatypes, although that single file may be compound (e.g. application/zip).
- Implementations with specific pen/digitizer hardware should (and implementations with other pointing devices may) support *scribble*—allowing in-place creation of pen-based data.
- Implementations with specific audio recording capabilities should support *record audio*—in-place recording of an audio clip.
- Implementations with a digital camera/scanner interface or screen capture should support *acquire image*—in-place upload of images from an attached device.
- Implementations with video recording capability should provide a *record video* option.
- Implementations with 3d capabilities should provide a 3d interface option.
- Implementations may provide proprietary implementations (for example, a mediaType of text/rtf could invoke an edit window with a proprietary word processing application)
- Implementations are encouraged to support other input devices not mentioned here.
- Implementations which cannot support upload for the given mediaType must make this apparent to the user.

Example: XML Representation: <upload>

```
<upload
  (single node binding attributes)
  (common attributes)
  mediaType = list of content types
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</upload>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**

common attributes defined in **8.12.1 Common Attributes**

mediaType - list of suggested media types, used by the XForms Processor to determine which input methods apply.

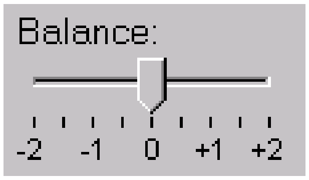
8.6 range

Description: This form control allows selection from a continuous range of values.

Example: Picking From A Range

```
<range ref="/stats/balance" start="-2.0" end="2.0" stepSize="0.5">
  <caption>Balance</caption>
</range>
```

A graphical browser might render this as follows:



Data Binding Restrictions: Binds only the following list of datatypes, or datatypes derived by restriction from those in the list: `xsd:duration`, `xsd:date`, `xsd:time`, `xsd:dateTime`, `xsd:gYearMonth`, `xsd:gYear`, `xsd:gMonthDay`, `xsd:gDay`, `xsd:gMonth`, `xsd:float`, `xsd:decimal`, `xsd:double`.

Implementation Requirements: Must allow input of a value corresponding to the bound datatype. Implementations should inform the user of the upper and lower bounds, as well as the step size, if any. In graphical environments, this form control may be rendered as a "slider" or "rotary control".

Notice that the attributes of this element encapsulate sufficient metadata that in conjunction with the type information available from the XForms Model proves sufficient to produce meaningful prompts when using modalities such as speech, e.g., when using an accessibility aid. Thus, an aural user agent might speak a prompt of the form *Please pick a date in the range January 1, 2001 through December 31, 2001.*

Example: XML Representation: `<range>`

```
<range
  (single node binding attributes)
  (common attributes)
  start = datavalue
  end = datavalue
  stepSize = datavalue-difference
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</range>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
common attributes defined in **8.12.1 Common Attributes**
start - optional hint for the lexical starting bound for the range—a legal value for the underlying data.
end - optional hint for the ending bound for the range—a legal value for the underlying data.
stepSize - optional hint to use for incrementing or decrementing the value. Should be of a type capable of expressing the difference between two legal values of the underlying data.

8.7 button

Description: This form control is similar to the HTML element of the same name and allows for user-triggered actions. This form control may also be used to advantage in realizing other custom form controls.

Example: Simple Button

```
<button>
  <caption>Click here</caption>
</button>
```

Data Binding Restrictions: Binding not possible for this form control.

Implementation Requirements: The user agent must provide a means to generate an `xforms:activate` event on the form control. Graphical implementations would typically render this form control as a push-button with the caption on the button face. Stylesheets can be used to style the button as an image.

Example: XML Representation: `<button>`

```
<button
  (common attributes)
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</button>
```

common attributes defined in **8.12.1 Common Attributes**

8.8 submit

Description: This form control initiates submission of all or part of the instance data to which it is bound.

Example: Submit

```
<submit submitInfo="timecard">
  <caption>Submit Timecard</caption>
</submit>
```

Data Binding Restrictions: Binding not possible for this form control.

Implementation Requirements: Upon receiving event `xforms:activate`, this form control dispatches event `xforms:submit` to the `submitInfo` element specified by required attribute `submitInfo`. If not specified, the first `submitInfo` element in document order is used.

Example: XML Representation: `<submit>`

```
<submit
  (common attributes)
  submitInfo = xsd:IDREF #REQUIRED
>
  <!-- caption, (help|hint|alert|action|extension)* -->
</submit>
```

submitInfo - Required reference to element `submitInfo`
common attributes defined in **8.12.1 Common Attributes**

8.9 selectOne

Description: This form control allows the user to make a single selection from multiple choices.

Example: Pick A Flavor

```
<selectOne ref="my:icecream/my:flavor">
  <caption>Flavour</caption>
  <item>
    <caption>Vanilla</caption>
    <value>v</value>
  </item>
  <item>
    <caption>Strawberry</caption>
    <value>s</value>
  </item>
  <item>
    <caption>Chocolate</caption>
    <value>c</value>
  </item>
</selectOne>
```

In the above example, selecting one of the choices will result in the associated value given by element `value` on the selected item being set in the underlying data instance at the location `icecream/flavor`.

A graphical browser might render this form control as any of the following:

Data Binding Restrictions: Binds to any simpleContent.

Implementation Requirements: The caption for each choice must be presented, allowing at all times exactly one selection. This form control stores the value corresponding to the selected choice in the location addressed by attribute `ref`. The value to be stored is either directly specified as the contents of element `value`, or specified indirectly through attribute `ref` on element `value`.

Note that the datatype bound to this form control may include a non-enumerated value space, e.g., `xsd:string`. In this case, control `selectOne` may have attribute `selection="open"`. The form control should then allow free data entry, as described in **8.1 input**.

For closed selections:If the initial instance value matches the storage value of one of the given items, that item is selected. If there is no match, the first item is initially selected.

For open selections: If the initial instance value matches the storage value specified by one of the items, the first such matching item is selected. Otherwise, the selected value is the initial lexical value. Free entry text is handled the same as form control `input` **8.1 input**.

User interfaces may choose to render this form control as a pulldown list or group of radio buttons, among other options. The selectUI attribute offers a hint as to which rendering might be most appropriate, although any styling information (such as CSS) should take precedence.

Typically, a stylesheet would be used to determine the exact appearance of form controls, though a means is provided to suggest an appearance through attribute `selectUI`. The value of the attribute consists of one of the following values, each of which may have a platform-specific look and feel.

radio
checkbox
menu
listbox

Example: XML Representation: `<selectOne>`

```
<selectOne
  (single node binding attributes)
  (common attributes)
  selectUI = ("radio" | "checkbox" | "menu" | "listbox" )
  selection = "open" | "closed" : "closed"
>
  <!-- caption, (choices|item|itemset)+, (help|hint|alert|action|extension)* -->
</selectOne>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
common attributes defined in **8.12.1 Common Attributes**
selectUI - appearance override
selection - optional attribute determining whether free entry is allowed in the list.

8.10 selectMany

Description: This form control allows the user to make multiple selections from a set of choices.

Example: Selecting Ice Cream Flavor

```
<selectMany ref="my:icecream/my:flavors">
  <caption>Flavours</caption>
  <choices>
    <item>
      <caption>Vanilla</caption>
      <value>v</value>
    </item>
    <item>
      <caption>Strawberry</caption>
      <value>s</value>
    </item>
    <item>
      <caption>Chocolate</caption>
      <value>c</value>
    </item>
  </choices>
</selectMany>
```

In the above example, more than one flavor can be selected.

A graphical browser might render form control `selectMany` as any of the following:

Data Binding Restrictions: any simpleContent capable of holding a sequence.

Note:

A limitation of the Schema list datatypes is that whitespace characters in the storage values (the `value="..."` attribute of the `item` element) are always interpreted as separators between individual data values. Therefore, authors should avoid using whitespace characters within storage values with list simpleContent.

Example: Incorrect Type Declaration

```
<item>
  <value>United States of America</value>
  ...
</item>
```

When selected, this item would introduce not one but four additional selection values: "America", "of", "States", and "United".

Implementation Hints: An accessibility aid might allow the user to browse through the available choices and leverage the grouping of choices in the markup to provide enhanced navigation through long lists of choices.

Example: XML Representation: `<selectMany>`

```
<selectMany
  (single node binding attributes)
  (common attributes)
  selectUI = ("radio" | "checkbox" | "menu" | "listbox")
>
  <!-- caption, (choices|item|itemset)+, (help|hint|alert|action|extension)* -->
</selectMany>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
common attributes defined in **8.12.1 Common Attributes**

selectUI - appearance override

8.11 Common Markup for selection controls

8.11.1 choices

This element is used within selection form controls to group available choices. This provides the same functionality as element `optgroup` in HTML.

Example: XML Representation: <choices>

```
<choices>
  <!-- caption?, (choices|item|itemset)+ -->
</choices>
```

8.11.2 item

This element specifies the storage value and caption to represent an item in a list. It is found within elements `selectOne` and `selectMany`, or grouped in element `choices`.

Example: XML Representation: <item>

```
<item>
  <!-- caption, value, (help|hint|alert|action|extension)* -->
</item>
```

id = xsd:ID - optional unique identifier.

8.11.3 itemset

Element `itemset` allows the creation of dynamic selections within controls `selectOne` and `selectMany`, where the available choices are determined at run-time. The node-set that holds the available choices is specified via attribute `nodeset`. Child elements `caption` and `value` indirectly specify the caption and storage values. Notice that the run-time effect of `itemset` is the same as using element `choices` to statically author the available choices.

Example: XML Representation: <itemset>

```
<itemset
  (node-set binding attributes)
>
  <!-- caption, value, (help|hint|alert|action|extension)* -->
</itemset>
```

node-set binding attributes - required node-set selector that specifies the node-set holding the available choices.

The following example shows element `itemset` within control `selectMany` to specify a dynamic list of ice cream flavors:

Example: Dynamic Choice Of Ice Cream Flavors

```
<model id="cone">
  <instance>
    <my:icecream>
      <my:flavours/>
    </my:icecream>
  </instance>
</model>
<model id="flavours">
  <instance>
    <my:flavours>
      <my:flavour type="v">
```

```
        <my:description>Vanilla</my:description>
      </my:flavour>
      <my:flavour type="s">
        <my:description>Strawberry</my:description>
      </my:flavour>
      <my:flavour type="c">
        <my:description>Chocolate</my:description>
      </my:flavour>
    </my:flavours>
  </instance>
</model>
<!-- user interaction markup -->
<selectMany model="cone" ref="my:icecream/my:flavours">
  <caption>Flavors</caption>
  <itemset model="flavours" nodeset="my:flavours/my:flavour">
    <caption ref="my:description"/>
    <value ref="@type"/>
  </itemset>
</selectMany>
```

8.11.4 value

This element provides a storage value to be used when an `item` is selected.

Data Binding Restriction: All lexical values must be valid according to the datatype bound to the selection control.

Example: XML Representation: <`value`>

```
<value
  (single node binding attributes)
>
  <!-- ##any -->
</value>
```

single node binding attributes - optional binding selector that specifies a location from where the storage value is to be fetched.

If inline content and a `ref` attribute are both specified, the `ref` attribute is used.

8.12 Common Markup

The preceding form control definitions make reference to child elements and attributes that are common to several of the form controls. This section defines these common markup components.

8.12.1 Common Attributes

The following attributes are common to many user-interface related XForms elements.

Example: XML Representation: Common Attributes

```
xml:lang = xsd:language
class = space separated list of classes
navIndex = xsd:nonNegativeInteger : 0
accessKey = xsd:token
```

xml:lang - Optional standard XML attribute to specify a human language for this element.
class - Optional selector for a style rule.
navIndex - Optional attribute is a non-negative integer in the range of 0-32767 used to define the navigation sequence. This gives the author control over the sequence in which form controls are traversed. The default navigation order is specified in the chapter **4 Processing Model**.

accessKey - Optional attribute defines a shortcut for moving the input focus directly to a particular form control. The value of this is typically a single character which when pressed together with a platform specific modifier key (e.g., the *alt* key) results in the focus being set to this form control.

8.12.2 Single Node Binding Attributes

The following attributes define a binding between a form control and an instance data node.

Example: XML Representation: Single Node Binding Attributes

```
ref = binding-expression
model = xsd:IDREF
bind = xsd:IDREF
```

ref - Binding expression. Details in the chapter **6 Constraints**. The first-node rule applies to the nodeset selected here.

model - Optional instance data selector. Details in the section **6.4.3 Binding References**.

bind - Optional reference to a bind element

It is an error if the `model` idref value refers to an id not on a `model` element, or if the `bind` idref value refers to an id not on a `bind` element.

8.12.3 Nodeset Binding Attributes

The following attributes define a binding between a form control and a node-set returned by the XPath expression.

Example: XML Representation: Nodeset Binding Attributes

```
nodeset = binding-expression
model = xsd:IDREF
bind = xsd:IDREF
```

nodeset - Binding expression. Details in the chapter **6 Constraints**.

model - Optional instance data selector. Details in the chapter **6 Constraints**.

bind - Optional reference to a bind element

It is an error if the `model` idref value refers to an id not on a `model` element, or if the `bind` idref value refers to an id not on a `bind` element.

8.12.4 Common Child Elements

The child elements detailed below provide the ability to attach metadata to form controls.

Instead of supplying such metadata e.g., the label for a form control as inline content of the contained element `caption`, the metadata can be pointed to by using a simple XLink attribute `xlink:href` on these elements. Notice that systematic use of this feature can be exploited in internationalizing XForms user interfaces by:

- Factoring all human readable messages to a separate resource XML file.
- Using URIs into this XML resource bundle within individual caption elements
- Finally, an XForms processor can use content negotiation to obtain the appropriate XML resource bundle, e.g., based on the `accept-language` headers from the client, to serve up the user interface with messages localized to the client's locale.

8.12.4.1 caption

The required element `caption` labels the containing form control with a descriptive label. Additionally, the caption makes it possible for someone who can't see the form control to obtain a short description while navigating between form controls.

Example: XML Representation: <caption>

```
<caption
  (common attributes)
  (single node binding attributes)
  xlink:href = xsd:anyURI
>
  <!-- ##any -->
</caption>
```

common attributes - defined in **8.12.1 Common Attributes**
single node binding attributes - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
xlink:href = xsd:anyURI - link to external caption

The caption specified can exist in instance data, in a remote document, or as inline text. If multiple captions are specified in this element, the order of preference is: `ref`, `xlink:href`, inline.

An accessibility aid would typically speak the metadata encapsulated here when the containing form control gets focus.

8.12.4.2 help

The optional element `help` provides a convenient way to attach help information to a form control. This is equivalent to a `xforms:help` event handler that responds with a `<message type="modeless">`.

Example: XML Representation: <help>

```
<help
  (common attributes)
  (single node binding attributes)
  xlink:href = xsd:anyURI
>
  <!-- ##any -->
</help>
```

(common attributes) - defined in **8.12.1 Common Attributes**
single node binding attributes - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
xlink:href = xsd:anyURI - link to external help

The message specified can exist in instance data, in a remote document, or as inline text. If multiple captions are specified in this element, the order of precedence is: `ref`, `xlink:href`, inline.

8.12.4.3 hint

The optional element `hint` provides a convenient way to attach hint information to a form control. This is equivalent to a `xforms:hint` event handler that responds with a `<message type="ephemeral">`.

Example: XML Representation: <hint>

```
<hint
  (common attributes)
  (single node binding attributes)
  xlink:href = xsd:anyURI
>
  <!-- ##any -->
</hint>
```

(common attributes) - defined in **8.12.1 Common Attributes**
single node binding attributes - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**

xlink:href = xsd:anyURI - link to external hint

The message specified can exist in instance data, in a remote document, or as inline text. If multiple captions are specified in this element, the order of precedence is: `ref`, `xlink:href`, inline.

8.12.4.4 alert

The optional element `alert` provides a convenient way to attach alert or error information to a form control. This is equivalent to a `xforms:alert` event handler that responds with a `<message type="modal">`.

Example: XML Representation: `<alert>`

```
<alert
  (common attributes)
  (single node binding attributes)
  xlink:href = xsd:anyURI
>
  <!-- ##any -->
</alert>
```

(common attributes) - defined in **8.12.1 Common Attributes**
single node binding attributes - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**
xlink:href = xsd:anyURI - link to external alert

The message specified can exist at in instance data, in a remote document, or as inline text. If multiple captions are specified in this element, the order of precedence is: `ref`, `xlink:href`, inline.

8.12.4.5 extension

Optional element `extension` is a container for application-specific extension elements from any namespace other than the XForms namespace. This specification does not define the processing of this element.

Example: XML Representation: `<extension>`

```
<extension>
  <!-- ##other -->
</extension>
```

For example, RDF metadata could be attached to an individual form control as follows:

```
<input ref="dataset/user/email" id="email-input">
  <caption>Enter your email address</caption>
  <extension>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about="#email-input">
      <my:addressBook>personal</my:addressBook>
      </rdf:Description>
    </rdf:RDF>
  </extension>
</input>
```

9 XForms User Interface

This chapter covers XForms features for combining form controls into user interfaces.

All form controls defined in **8 Form Controls** are treated as individual units for purposes of visual layout e.g., in XHTML processing. Aggregation of form controls with markup defined in this chapter provides semantics about the relationship among user interface controls; such knowledge can be useful in delivering a coherent UI to small devices. For example, if the user interface needs to be split up over several screens, controls appearing inside the same aggregation would typically be rendered on the same screen or page.

9.1 group

The `group` element is used as a container for defining a hierarchy of form controls. Groups can be nested to create complex hierarchies.

Example: XML Representation: <group>

```
<group
  (single node binding attributes)
  (common attributes)
  >
  <!-- caption?, ##all -->
</group>
```

(single node binding attributes) - Selection of instance data node, defined in **8.12.2 Single Node Binding Attributes**

(common attributes) - defined in **8.12.1 Common Attributes**

Example: Grouping Related Controls

```
<group ref="address">
  <caption>Shipping Address</caption>
    <input ref="line_1">
      <caption>Address line 1</caption>
    </input>
    <input ref="line_2">
      <caption>Address line 2</caption>
    </input>
    <input ref="postcode">
      <caption>Postcode</caption>
    </input>
</group>
```

The hierarchy defined by nested group elements is used to determine the traversal order specified by attribute `navIndex` on form controls. Setting the input focus on a group results in the focus being set to the first form control in the tabbing order within that group.

9.2 switch

The XForms model allows the authoring of dynamic user interfaces that vary based on the current state of the instance data being populated—see model item property `relevant` **6.1.4 relevant**. As an example, portions of a questionnaire pertaining to the user's automobile may become *relevant* only if the user has answered in the affirmative to the question 'Do you own a car?'. Another example is when the underlying XForms model contains conditional structures.

This section defines construct `switch` that allows the creation of user interfaces where the user interface can be varied based on user actions and events. Element `switch` contains one or more `case` elements. Markup contained within element `case` specifies the user interface that is displayed when that `case` is *selected*. Boolean attribute `selected` of element `case` determines the *selected state* and can be manipulated programmatically via the DOM, or declaratively via XForms action `toggle`. Attribute `id` of `case` is used within action `toggle` for this purpose. The following example demonstrates the use of `switch`.

Example: switch

```
<switch id="sw">
  <case id="in" selected="true">
    <input ref="yourname">
      <caption>Please tell me your name</caption>
      <action ev:event="activate">
        <toggle case="out"/>
      </action>
    </input>
  </case>
  <case id="out" selected="false">
    <html:p>Hello <output ref="yourname" />.
      <button id="editButton">
        <caption>Edit</caption>
        <action id="editAction" ev:event="activate">
          <toggle case="in"/>
        </action>
      </button>
    </html:p>
  </case>
</switch>
```

The above results in the portion of the user interface contained in the first `case` being displayed initially. This prompts for the user's name; filling in a value and *activating* the control e.g., by pressing `return` results in XForms event `activate`. Event `activate` is handled by the attached handler—element `action`. This handler uses XForms action `toggle` to select the `case` with `id="out"`. This sets attribute `selected` on `case id="out"` to `true`. The markup contained in the selected `case` displays the name the user entered along with an `edit` button. Activating the edit button in turn results in the attached handler selecting `case id="in"`.

Example: XML Representation: <`switch`>

```
<switch
  (common attributes)
>
  <!-- case+ -->
</switch>
```

(common attributes) - defined in **8.12.1 Common Attributes**

Example: XML Representation: <case>

```
<case
  selected = xsd:boolean
>
  <!-- ##any -->
</case>
```

selected = xsd:boolean - optional selection status for the case.

If multiple `case`s within a `switch` are marked as `selected="true"`, the first selected `case` remains and all others are deselected. If none are selected, the first becomes selected.

9.3 repeat

The XForms Model allows the definition of repeating structures such as multiple items within a purchase order. When defining the XForms Model, such higher-level collections are constructed out of basic building blocks; similarly, this section defines user interface construct `repeat` that can bind to data structures such as lists and collections. Element `repeat` encapsulates a UI mapping over a homogeneous collection, in other words, a collection consisting of data items having the same type and structure.

Example: Shopping Cart

```
<repeat nodeset="/cart/items/item">
  <input ref="." .../><html:br/>
</repeat>
```

Element `repeat` operates over a homogeneous collection by *binding* the encapsulated user interface controls to each element of the collection. Attributes on element `repeat` specify how many members of the collection are presented to the user at any given time. XForms actions `insert`, `delete` and `setRepeatCursor` can be used to operate on the collection—see **10 XForms Actions**. Another way to view repeat processing (disregarding special user interface interactions) is to consider "unrolling" the repeat. The above example is similar to the following (assuming four item elements in the returned node-set):

Example: Repeat Unrolled

```
<!-- unrolled repeat -->
  <input ref="/cart/items/item[1]" .../><html:br/>
  <input ref="/cart/items/item[2]" .../><html:br/>
  <input ref="/cart/items/item[3]" .../><html:br/>
  <input ref="/cart/items/item[4]" .../><html:br/>
```

Notice that the model for the collection being populated would typically have defined attributes `minOccurs` and `maxOccurs`; these values may in turn determine if the user agent displays appropriate UI controls for the user to add or delete entries in the collection.

Example: Homogeneous Collection

```
<model>
  <instance>
    <my:lines>
      <my:line name="a">
        <my:price>3.00</my:price>
      </my:line>
      <my:line name="b">
        <my:price>32.25</my:price>
      </my:line>
      <my:line name="c">
        <my:price>132.99</my:price>
      </my:line>
    </my:lines>
  </instance>
</model>
      ...
<repeat id="lineset" nodeset="my:lines/my:line">
  <input ref="my:price">
    <caption>Line Item</caption>
  </input>
  <input ref="@name">
    <caption>Name</caption>
  </input>
</repeat>

<button>
  <caption>Insert a new item after the current one</caption>
  <action ev:event="activate">
    <insert nodeset="my:lines/my:line"
            at="cursor('lineset')"
            position="after"/>
    <setValue ref="my:lines/my:line[cursor('lineset')]/@name"/>
    <setValue ref="my:lines/my:line[cursor('lineset')]/price">0.00</setValue>
  </action>
</button>

<button>
  <caption>remove current item</caption>
  <delete ev:event="activate" nodeset="my:lines/my:line"
          at="cursor('lineset')"/>
</button>
```

Example: XML Representation: `<repeat>`

```
<repeat
  (node-set binding attributes)
  (common attributes)
  startIndex = xsd:positiveInteger : 1
  number = xsd:nonNegativeInteger
>
  <!-- ##any -->
</repeat>
```

(nodeset binding attributes) - Selection of context node-set, defined in **8.12.3 Nodeset Binding Attributes**
(common attributes) - defined in **8.12.1 Common Attributes**
startIndex - 1-based hint to the XForms Processor as to which starting element from the collection to display.
number - hint to the XForms Processor as to how many elements from the collection to display.

9.3.1 Repeat Processing

The markup contained within the body of element `repeat` specifies the user interface to be generated for each member of the underlying collection. During user interface initialization (see **4.2.4 xforms:UIInitialize**), the following steps are performed for `repeat`:

1. Attribute `nodeset` is evaluated to locate the homogeneous collection to be operated on by this `repeat`.
2. The corresponding nodes in element `instance` in the source document are located—these nodes provide initial values and also serve as a *prototypical instance* for constructing members of the repeating collection.
3. The *cursor* for this repeating structure is initialized to point at the head of the collection.
4. The user interface template specified within element `repeat` is *bound* to this prototypical instance. If there is a type mismatch between the prototypical instance and the binding restrictions for the user interface controls, an error is signaled and processing stops.
5. User interface as specified by the `repeat` is generated for the requisite number of members of the collection as specified by attributes on element `repeat`, and model item constraints `minOccurs` and `maxOccurs`.

The user interface markup for repeating structures adds encapsulation metadata about the collection being populated. The processing model for repeating structures uses a *cursor* that points to the *current* item in the data instance. This repeat cursor is accessed via XForms extension functions `cursor` **7.4.2.5 cursor()** and manipulated via XForms action `setRepeatCursor` **10.10 setRepeatCursor**. This cursor is used as a reference point for `insert` and `delete` operations. Notice that the contained XForms form controls inside element `repeat` do not explicitly specify the index of the collection entry being populated. This is intentional; it keeps both authoring as well as the processing model simple.

The binding expression attached to the repeating sequence returns a node-set of the collection being populated, not an individual node. Within the body of element `repeat`, binding expressions are evaluated with a context node of the node determined by the repeatCursor. Repeat processing uses XPath expressions to address the collection over which element `repeat` operates. The XPath expression used as the value of attribute `nodeset` must select a node-set of contiguous child element nodes, with the same local name and namespace name of a common parent node—this ensures that the node-set represent a homogeneous collection. The behavior of element `repeat` with respect to other XPath node-sets is undefined. The initial instance data supplies the prototypical member of the homogeneous collection, and this is used during UI initialization—**4.2.4 xforms:UIInitialize**—to construct the members of the homogeneous collection. This prototypical instance is also used by action `insert` when creating new members of the collection. To create homogeneous collections, the initial instance data *must* specify at least one member of the collection; this requirement is similar to *requiring* instance data in addition to a schema, and the same justification applies.

The form controls appearing inside `repeat` need to be suitable for populating individual items of the collection. A simple but powerful consequence of the above is that if the XForms Model specifies nested collections, then a corresponding user interface can nest `repeat` elements.

9.3.2 Nested Repeats

It is possible to nest repeat elements to create more powerful user interface for editing structured data. **E.2 Editing Hierarchical Bookmarks Using XForms** is an example of a form using nested repeats to edit hierarchical data consisting of bookmarks within multiple sections. Notice that an inner repeat's cursor always starts from `1`. Consider the following `insert` statement that appears as part of that example.

Example: Repeat Cursor And Nested Repeats

```
<xforms:insert
  nodeset="/bookmarks/section[cursor('repeatSections')]/bookmark"
  at="cursor('repeatBookmarks')"
  position="after"/>
```

The above `insert` statement is used in that example to add new bookmark entries into the *currently selected* section. The inner (nested) repeat operates on bookmarks in this selected section; The cursor—as returned by XForms function `cursor`—for this inner repeat starts at `1`. Hence, after a new empty section of bookmarks is created and becomes *current*, the first *insert bookmark* operation adds the newly created bookmark at the front of the list.

9.3.3 User Interface Interaction

Element `repeat` enables the binding of user interaction to a homogeneous collection. The number of displayed items might be less than the total number available in the collection. In this case, the presentation would render only a portion of the repeating items at a given time. For example, a graphical user interface might present a scrolling table. The current item indicated by the repeat cursor should be presented at all times, for example, not allowed to scroll out of view. The XForms Actions enumerated at **10 XForms Actions** may be used within event listeners to manipulate the homogeneous collection being populated by scrolling, inserting and deleting entries.

Notice that the markup encapsulated by element `repeat` acts as the template for the user interaction that is presented to the user. As a consequence, it is not possible to refer to portions of the generated user interface via statically authored `idref` attributes. A necessary consequence of this is that XForms 1.0 does not specify the behavior of construct `switch` within element `repeat`. Future versions of XForms may specify the behavior of `switch` inside `repeat` based on implementation experience and user feedback.

10 XForms Actions

All form controls defined in this specification have a set of common *behaviors* that encourage consistent authoring and look and feel for XForms-based applications. This consistency comes from attaching a common set of behaviors to the various form controls. In conjunction with the event binding mechanism provided by [XML Events], these handlers provide a flexible means for forms authors to specify event processing at appropriate points within the XForms user interface. XForms actions are declarative XML event handlers that capture high-level semantics. As a consequence, they significantly enhance the accessibility of XForms-based applications in comparison to previous web technologies that relied exclusively on scripting.

NOTE: This example is based on the XML Events specification [XML Events], which is proceeding independently from XForms, and thus might be slightly incorrect.

Example: Action Syntax

```
<xforms:button>
  <xforms:caption>Reset</xforms:caption>
  <xforms:resetInstance ev:event="xforms:activate"/>
</xforms:button>
```

This example recreates the behavior of the HTML *reset* button, which this specification does not define as an independent form control.

For each built-in XForms action, this chapter lists the following:

Name
Description of behavior
XML Representation
Sample usage

All elements defined in this chapter explicitly allow global attributes from the XML Events namespace, and apply the processing defined in that specification in section 2.3 [XML Events].

10.1 dispatch

This action dispatches an XForms Event to a specific element identified by the `target` attribute. Two kinds of event can be dispatched:

One of the predefined XForms events (i.e., xforms:event-name), in which case the bubbles and cancelable attributes are ignored and the standard semantics as defined in the Processing model apply.
An event created by the XForms author with no predefined XForms semantics and as such not handled by default by the XForms processor.

Example: XML Representation: `<dispatch>`

```
<dispatch
  name = xsd:NMTOKEN
  target = xsd:IDREF
  bubbles = xsd:boolean : true
  cancelable = xsd:boolean : true
/>
```

name = xsd:NMTOKEN - required name of the event to dispatch.
target = xsd:IDREF - required reference to the event target.
bubbles = xsd:boolean : true - boolean indicating if this event bubbles—as defined in DOM2 events.
cancelable = xsd:boolean : true - boolean indicating if this event is cancelable—as defined in DOM2 events.

10.2 refresh

This action dispatches an `xforms:refresh` event. This action results in the XForms user interface being *refreshed*, and the presentation of user interface controls being updated to reflect the state of the underlying instance data --see **4.3.15 xforms:refresh**

Example: XML Representation: `<refresh>`

```
<refresh/>
```

10.3 recalculate

This action dispatches an `xforms:recalculate` event. As a result, instance data nodes whose values need to be recomputed are updated as specified in the processing model --see **4.3.17 xforms:recalculate**.

Example: XML Representation: `<recalculate>`

```
<recalculate/>
```

10.4 revalidate

This action dispatches an `xforms:revalidate` event. This results in the instance data being revalidated as specified by the processing model --see **4.3.16 xforms:revalidate**

Example: XML Representation: `<revalidate>`

```
<revalidate/>
```

10.5 setFocus

This action sets focus to the form control referenced by the `idref` attribute by dispatching an `xforms:focus` event. Note that this event is implicitly invoked to implement XForms accessibility features such as `accessKey`.

Example: XML Representation: `<setFocus>`

```
<setFocus
  idref = xsd:IDREF
/>
```

idref = xsd:IDREF - required reference to a form control

Setting focus to a repeating structure sets the focus to the member represented by the repeat cursor.

10.6 loadURI

This action traverses the specified XLink.

Example: XML Representation: `<loadURI>`

```
<loadURI
  (single node binding attributes)
  xlink:href = xsd:anyURI
  xlink:show = ("new" | "replace" | "embed" | "other" | "none")
/>
```

(single node binding attributes) - Selects the instance data node containing the URI.
xlink:href - optional URI to load.
xlink:show - optional link behavior specifier.

Either the single node binding attributes, pointing to a URI in the instance data, or the attribute `xlink:href` are required. If both are present, the action has no effect.

Possible values for attribute `xlink:show` have the following processing for the document (or portion of a document) reached by traversing the link:

new

The document is loaded into a new window (or other presentation context). Form processing in the original window continues.

replace

The document is loaded into the current window. Form processing is interrupted, exactly as if the user had manually requested navigating to a new document.

embed

The document is incorporated into the current window in an application-specific manner. Form processing continues.

other

The document is loaded in an application-specific manner. The application should look for other markup present in the link to determine the appropriate behavior.

none

The document is loaded in an application-specific manner. The application should *not* look for other markup present in the link to determine the appropriate behavior.

10.7 setValue

This action explicitly sets the value of the specified instance data node.

Example: XML Representation: <setValue>

```
<setValue
  (single node binding attributes)
  value = XPath expression
>
  <!-- literal value -->
</setValue>
```

(single node binding attributes) - Selects the instance data node where the value is to be stored.

value = XPath expression - XPath expression to evaluate, with the result stored in the selected instance data node.

The element content of `setValue` specifies the literal value to set; this is an alternative to specifying a computed value via attribute `value`. The following two examples contrast these approaches:

Example: setValue with Expression

```
<setValue bind="put-here" value="a/b/c"/>
```

This causes the string value at `a/b/c` in the instance data to be placed on the single node selected by the bind element with `id="put-here"`.

Example: setValue with Literal

```
<setValue bind="put-here">literal string</setValue>
```

This causes the value "literal string" to be placed on the single node selected by the bind element with `id="put-here"`.

If neither a `value` attribute nor text content are present, the effect is to set the value of the selected node to the empty string ("").

10.8 submitInstance

This action initiates submit processing by dispatching an `xforms:submit` event. Processing of event `xforms:submit` is defined in the processing model—see **4.4.1 xforms:submit**.

Example: XML Representation: <submitInstance>

```
<submitInstance
  submitInfo = xsd:IDREF />
```

id = xsd:ID - optional unique identifier.

submitInfo = xsd:IDREF - optional reference to a `submitInfo` element.

Note:

This XForms Action is a convenient way of expressing the following:

```
<dispatch target="mysubmitinfo" name="submitInstance"/>
```

10.9 resetInstance

This action initiates reset processing by dispatching an `xforms:reset` event to the specified `model`. Processing of event `xforms:reset` is defined in the processing model—see **4.3.18 xforms:reset**.

Example: XML Representation: <resetInstance>

```
<resetInstance
  model = xsd:IDREF
/>
```

model = xsd:IDREF - Selection of instance data for reset, defined in **8.12.3 Nodeset Binding Attributes**

10.10 setRepeatCursor

This action marks a specific item as current in a repeating sequence (within **9.3 repeat**).

Example: XML Representation:Action `<setRepeatCursor>`

```
<setRepeatCursor
  repeat = xsd:IDREF
  cursor = XPath expression that evaluates to number
/>
```

repeat = xsd:IDREF - required reference to a repeat
cursor = XPath expression that evaluates to number - required 1-based offset into the sequence.

10.11 insert

This action is used to insert new entries into a homogeneous collection, e.g., a set of items in a shopping cart. Attributes of action `insert` specify the insertion in terms of the collection in which a new entry is to be inserted, and the position within that collection where the new node will appear. The new node is created by cloning the final member of the homogeneous collection specified by the initialization instance data. In this process, nodes of type `xsd:ID` are not copied. The rules for insert processing are as follows:

1. The homogeneous collection to be updated is determined by evaluating binding attribute `nodeset`.
2. The corresponding node-set of the initial instance data is located to determine the prototypical member of the collection. The final member of this collection is cloned to produce the node that will be inserted. Finally, this newly created node is inserted into the instance data at the position specified by attributes `position` and `at`.

   Attribute `at` is evaluated to determine the insertion index—a numerical value that is the index into the node-set. Attribute `position` specifies whether the new node is inserted *before* or *after* this index.

   The rules for selecting the index are as follows:

   1. The return value of the XPath expression in attribute `at` is processed according to the rules of the XPath function `round()`. For example, the literal `1.5` becomes `2`, and the literal `'string'` becomes `NaN`.
   2. If the result is `NaN`, the insert operation has no effect.
   3. If the result is not a valid index for the node-set, it is clipped to either `1` or the size of the node-set, whichever is closer.
3. Finally, the cursor for any `repeat` that is bound to the homogeneous collection where the node was added is updated to point to the newly added node.

This action results in the insertion of newly created data nodes into the XForms data instance. Such nodes are constructed as defined in the initialization section of the processing model—see **4.2 Initialization Events**. Following the insertion of the newly created node into the instance data, events `xforms:recalculate`, `xforms:revalidate` and `xforms:refresh` are triggered in sequence. As an example, this causes the instantiation of the necessary user interface for populating a new entry in the underlying collection when used in conjunction with repeating structures **9.3 repeat**.

Example: XML Representation:Action `<insert>`

```
<insert
  (node-set binding attributes)
  at = XPath expression
  position = "before" | "after"
/>
```

(nodeset binding attributes) - Selection of instance data nodes, defined in **8.12.3 Nodeset Binding Attributes**
at - required XPath expression evaluated to determine insert location.

position - required selector if insert before/after behavior.

An example of using `insert` with a repeating structure is located at **9.3 repeat**. Note that XForms Action `setValue` can be used in conjunction with `insert` to provide initial values for the newly inserted nodes.

10.12 delete

This action deletes nodes from the instance data. The rules for delete processing are as follows:

1. The homogeneous collection to be updated is determined by evaluating binding attribute `nodeset`. If the collection is empy, the delete action has no effect.
2. The `n`-th node is deleted from the instance data, where `n` represents the number returned from node-set index evaluation, defined in **10.11 insert**.
3. If the last item in the collection is removed, the cursor position becomes 0. Otherwise, the cursor will point to the new `n`-th item.

This action results in deletion of nodes in the instance data. Following the specified deletion, events `xforms:recalculate`, `xforms:revalidate` and `xforms:refresh` are triggered in sequence. As an example, this causes the destruction of the necessary user interface for populating a deleted entry in the underlying collection when used in conjunction with repeating structures **9.3 repeat**.

Example: XML Representation:Action `<delete>`

```
<delete
  (node-set binding attributes)
  at = XPath expression
/>
```

(nodeset binding attributes) - Selection of instance data nodes, defined in **8.12.3 Nodeset Binding Attributes**
at - XPath expression evaluated to determine insert location.

An example of using `delete` with a repeating structure is located at **9.3 repeat**.

10.13 toggle

This action selects one possible case from an exclusive list of choices e.g., encapsulated by `switch` see **9.2 switch**, by:

1. Dispatching an `xforms:deselect` event to the currently selected item.
2. Dispatching an `xform:select` event to the item to be selected.

Example: XML Representation: Action `<toggle>`

```
<toggle
  case = xsd:IDREF
/>
```

case = xsd:IDREF - required reference to a case section inside the conditional construct

The `toggle` action adjusts all `selected` attributes on the affected `case`s to reflect the new state.

10.14 script

This action encapsulates an event handler authored in the specified scripting language. The handler may be *inline*, i.e., as PCDATA content of element `script`; alternatively it may be contained in an external resource and referred to via XML-events attribute `ev:handler`. Optional attribute `role` serves as documentation for the handler.

Example: XML Representation: Action `<script>`

```
<script
  type = xsd:string
  role=xsd:string
>
  <!-- #CDATA -->
</script>
```

type = xsd:string - required mime-type identifier of scripting language.
role = xsd:string - Optional descriptive text documenting the contained script.

10.15 message

This action encapsulates a message to be displayed to the user.

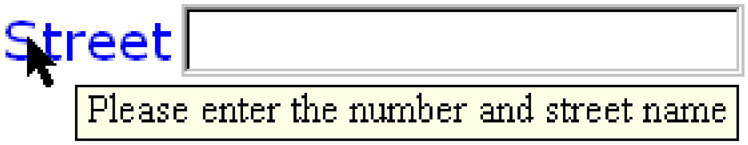
Example: XML Representation: <message>

```
<message
  (single node binding attributes)
  xlink:href = xsd:anyURI
  level = "ephemeral" | "modeless" | "modal"
>
  <!-- mixed content -->
</message>
```

(single node binding attributes) - optional attributes that point to the instance data for a string message.
xlink:href = xsd:anyURI - optional specifier of an external resource for the message.
level - required message level identifier.

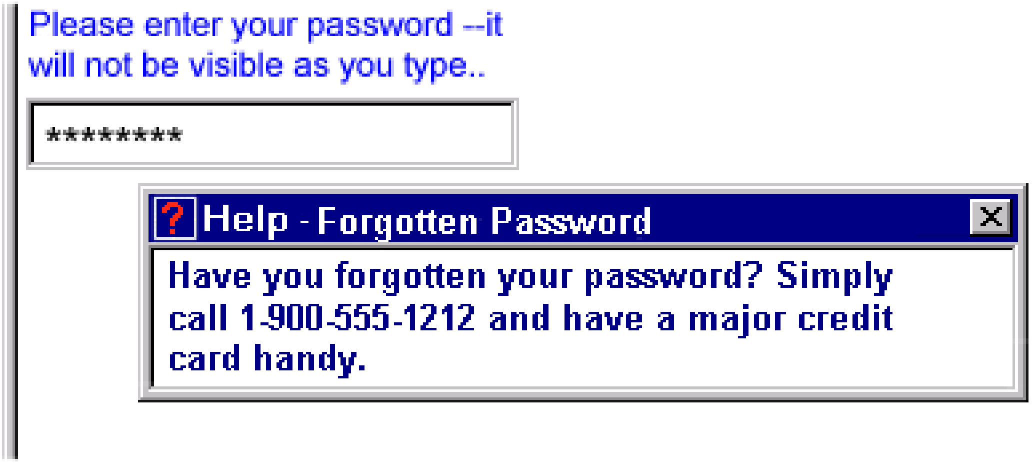
The message specified can exist in instance data, in a remote document, or as inline text. If multiple captions are specified in this element, the order of preference is: `ref`, `xlink:href`, inline.

A graphical browser might render an ephemeral message as follows:



Please enter the number and street name

A graphical browser might render a modeless message as follows:

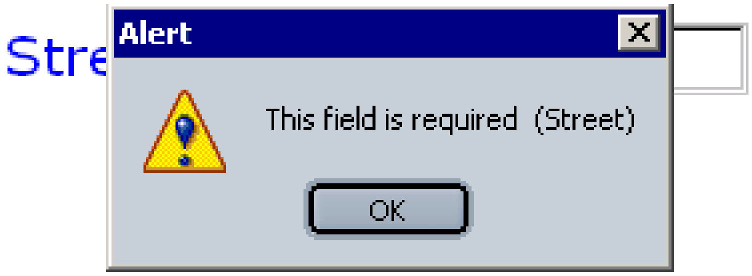


Please enter your password --it will not be visible as you type..

****

Have you forgotten your password? Simply call 1-900-555-1212 and have a major credit card handy.

A graphical browser might render a modal message as follows:



This field is required (Street)

10.16 action

Action `action` is used to group multiple actions.

Example: XML Representation: <`action`>

```
<action
>
  <!-- Action handlers -->
</action>
```

When using element `action` to group actions, care should be taken to list the event on element `action`, rather than on the contained actions.

Example: Grouping Actions

```
<button>
  <caption>Click me</caption>
  <action ev:event="xforms:activate">
    <resetInstance/>
    <setValue/>
  </action>
</button>
```

Notice that in the above example, `ev:event="xforms:activate"` occurs on element `action`. Placing `ev:event="xforms:activate"` on either or both of the contained actions will have no effect. This is because the above example relies on the defaulting of XML-Event attributes `observer` and `handler`. As defined in the XML-Events specification, if both observer and handler attributes are omitted, then the parent is the observer. Placing `ev:event="xforms:activate"` on the children of element `action` therefore causes element `action` to become the *observer* for the individual actions. Consequently, these actions will never be triggered since events arrive at element `button`, not element `action`.

11 Conformance

11.1 Conformance Levels

The XForms specification is designed for implementation on hardware platforms of all sizes, from tiny hand-held devices to high-powered servers. Clearly, a one-size-fits-all approach has its drawbacks. For this reason, there are two conformance levels for XForms Processors, documents, and authoring tools.

11.1.1 XForms Basic

This conformance level is suitable for devices with limited computing power, such as mobile phones, hand-held computers, and appliances. This conformance level uses a subset of XML Schema and does not include any resource-intensive features. Resource-limited XForms Processors may define implementation limits on the maximum size of a `node-set` returned by XPath expressions. **XForms Basic** implementations should return "`basic`" for the `conformance-level` property.

11.1.2 XForms Full

This conformance level is suitable for more powerful forms processing, such as might be found on a standard desktop browser or a server. **XForms Full** implementations should return "`full`" for the `conformance-level` property.

11.2 Conformance Description

11.2.1 Conforming XForms Processors

- All XForms Processors must support the required portions of the specifications normatively listed as references (**B References**).
- XForms Basic Processors must implement all required features labeled **Basic**.
- XForms Full Processors must implement all required features.

11.2.2 Conforming XForms Documents

All XForms Containing Documents must conform to the required portions of the specifications normatively listed as references (**B References**). XForms elements are typically inserted into a containing document in multiple places. The root element for each individual fragment must be `model`, a form control, or one of `group`, `switch`, `repeat`. Individual XForms fragments must be schema-valid against the Schema for XForms (**A Schema for XForms**).

All XForms Basic conformant Documents must conform to all required portions of this specification marked as Basic, and additionally not include any features not specifically marked as Basic.

All XForms Full conformant Documents must conform to all required portions of this specification.

11.2.3 Conforming XForms Generators

XForms Basic Generators must create conforming XForms Basic documents.

XForms Full Generators must create conforming XForms Basic and XForms Full documents, depending on the author's choice.

12 Glossary Of Terms

Binding

[Definition: A "binding" connects an instance data node to a form control or to a model item constraint by using a binding expression as a locator.]

Binding expression

[Definition: An XPath LocationPath expression used in a binding.]

Computed expression

[Definition: An XPath expression used by model item properties such as relevant and calculate to include dynamic functionality in XForms.]

Containing document

[Definition: A specific document, for example an XHTML document, in which one or more <model> elements are found.]

Datatype

[Definition: From XML Schema [XML Schema part 2]: A 3-tuple, consisting of a) a set of distinct values, called its value space, b) a set of lexical representations, called its lexical space, and c) a set of facets that characterize properties of the value space, individual values or lexical items.]

Facet

[Definition: From XML Schema [XML Schema part 2]: A single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions.]

Form control

[Definition: An XForms user interface control that serves as a point of user interaction.]

Instance data

[Definition: An internal tree representation of the values and state of all the instance data nodes associated with a particular form.]

Instance data node

[Definition: An XPath node from the instance data.]

Lexical space

[Definition: From XML Schema [XML Schema part 2]: A lexical space is the set of valid literals for a datatype.]

Model item

[Definition: An instance data node with associated constraints.]

Model item constraint

[Definition: Either a Schema constraint or an XForms constraint.]

Schema constraint

[Definition: A restriction, applied to form data, based on XML Schema datatypes.]

Value space

[Definition: From XML Schema [XML Schema part 2]: A set of values for a given datatype. Each value in the value space of a datatype is denoted by one or more literals in its lexical space.]

XForms constraint

[Definition: A restriction, applied to form data, based on XForms-specific expressions.]

XForms Model

[Definition: The non-visible definition of an XML form as specified by XForms. The XForms Model defines the individual model items and constraints and other run-time aspects of XForms.]

XForms Processor

[Definition: A software application or program that implements and conforms to the XForms specification.]

A Schema for XForms

```
<?xml version="1.0"?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by Micah Dubinko (XForms WG) -->
<!--$Id: index-all.fo,v 1.4 2002/01/16 23:32:13 tvraman Exp $-->
<xsd:schema targetNamespace="http://www.w3.org/2002/01/xforms"
      xmlns:ev="http://www.w3.org/2001/xml-events"
                xmlns:xlink="http://www.w3.org/1999/xlink"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xforms="http://www.w3.org/2002/01/xforms"
                elementFormDefault="qualified">
  <!--
Open Issues

Need a datatype for 'list of mediaTypes' on mediaType of <upload>
-->
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.
    <xsd:annotation>
      <xsd:documentation>Get access to xml:lang and friends</xsd:documentation>
    </xsd:annotation>
  </xsd:import>
  <xsd:import namespace="http://www.w3.org/1999/xlink" schemaLocation="XLink-Schema.xsd"/>
  <xsd:import namespace="http://www.w3.org/2001/xml-events" schemaLocation="XML-Events-Sch
  <!--
structural elements
-->
  <xsd:attributeGroup name="horzAttrs">
    <xsd:annotation>
      <xsd:documentation>Attributes for _every_ element in XForms</xsd:documentation>
    </xsd:annotation>
    <xsd:anyAttribute namespace="##other"/>
  </xsd:attributeGroup>
  <xsd:element name="model">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xforms:instance" minOccurs="0"/>
        <xsd:element ref="xforms:schema" minOccurs="0"/>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:choice>
            <xsd:element ref="xforms:submitInfo"/>
            <xsd:element ref="xforms:privacy"/>
            <xsd:element ref="xforms:bind"/>
            <xsd:element ref="xforms:action"/>
            <xsd:element ref="xforms:extension"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="extensionFunctions" type="xforms:QNameList" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="schema">
    <xsd:annotation>
      <xsd:documentation>schema container.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##other"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="instance">
    <xsd:annotation>
      <xsd:documentation>instance container.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##any" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="privacy">
    <xsd:annotation>
      <xsd:documentation>privacy reference.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="submitInfo">
    <xsd:annotation>
      <xsd:documentation>submit info container.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:group ref="xforms:actionGroup"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="action" type="xsd:anyURI" use="optional"/>
      <xsd:attribute name="mediaTypeExtension" use="optional" default="none">
        <xsd:simpleType>
          <xsd:union memberTypes="xforms:QNameButNotNCNAME">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="none"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:union>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="method" use="optional" default="post">
        <xsd:simpleType>
          <xsd:union memberTypes="xforms:QNameButNotNCNAME">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="post"/>
                <xsd:enumeration value="get"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:union>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="version" type="xsd:NMTOKEN" use="optional"/>
      <xsd:attribute name="indent" type="xsd:boolean" use="optional"/>
      <xsd:attribute name="encoding" type="xsd:string" use="optional"/>
      <xsd:attribute name="mediaType" type="xsd:string" use="optional"/>
      <xsd:attribute name="omitXMLDeclaration" type="xsd:boolean" use="optional"/>
      <xsd:attribute name="standalone" type="xsd:boolean" use="optional"/>
      <xsd:attribute name="CDATASectionElements" type="xforms:QNameList" use="optional"/>
      <xsd:attribute name="replace" use="optional" default="all">
        <xsd:simpleType>
          <xsd:union memberTypes="xforms:QNameButNotNCNAME">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="all"/>
                <xsd:enumeration value="instance"/>
                <xsd:enumeration value="none"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:union>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <xsd:attributeGroup name="linkingAttributes">
    <xsd:attribute ref="xlink:type" default="simple"/>
    <xsd:attribute ref="xlink:href"/>
    <xsd:attribute name="href" type="xsd:anyURI" use="prohibited"/>
  </xsd:attributeGroup>
  <xsd:element name="bind">
    <xsd:annotation>
      <xsd:documentation>Definition of bind container.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="xforms:bind"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="ref" type="xsd:string" use="optional"/>
      <xsd:attribute name="type" type="xsd:QName" use="optional"/>
      <xsd:attribute name="readOnly" type="xsd:string" use="optional"/>
      <xsd:attribute name="required" type="xsd:string" use="optional"/>
      <xsd:attribute name="relevant" type="xsd:string" use="optional"/>
      <xsd:attribute name="isValid" type="xsd:string" use="optional"/>
      <xsd:attribute name="calculate" type="xsd:string" use="optional"/>
      <xsd:attribute name="maxOccurs" use="optional">
        <xsd:simpleType>
          <xsd:union memberTypes="xsd:nonNegativeInteger">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="unbounded"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:union>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="minOccurs" type="xsd:nonNegativeInteger" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--
User Interface form controls
-->
  <xsd:group name="formControls">
    <xsd:choice>
      <xsd:element ref="xforms:input"/>
      <xsd:element ref="xforms:textarea"/>
      <xsd:element ref="xforms:secret"/>
      <xsd:element ref="xforms:output"/>
      <xsd:element ref="xforms:upload"/>
      <xsd:element ref="xforms:selectOne"/>
      <xsd:element ref="xforms:selectMany"/>
      <xsd:element ref="xforms:range"/>
      <xsd:element ref="xforms:submit"/>
      <xsd:element ref="xforms:button"/>
    </xsd:choice>
  </xsd:group>
  <xsd:attributeGroup name="bindFirstAttributes">
    <xsd:attribute name="model" type="xsd:IDREF" use="optional"/>
    <xsd:attribute name="ref" type="xsd:string" use="optional"/>
    <xsd:attribute name="bind" type="xsd:IDREF" use="optional"/>
  </xsd:attributeGroup>
  <xsd:attributeGroup name="bindAllAttributes">
    <xsd:attribute name="model" type="xsd:IDREF" use="optional"/>
    <xsd:attribute name="nodeset" type="xsd:string" use="optional"/>
    <xsd:attribute name="bind" type="xsd:IDREF" use="optional"/>
  </xsd:attributeGroup>
  <xsd:attributeGroup name="commonUIAttributes">
    <xsd:attribute ref="xml:lang" type="xsd:language" use="optional"/>
    <xsd:attribute name="class" type="xsd:string" use="optional"/>
    <xsd:attribute name="accessKey" type="xsd:string" use="optional"/>
    <xsd:attribute name="navIndex" type="xsd:nonNegativeInteger" use="optional"/>
  </xsd:attributeGroup>
  <xsd:element name="caption">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="hint">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="help">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="alert">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
      <xsd:attributeGroup ref="xforms:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="extension">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##other"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="choices">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xforms:caption" minOccurs="0"/>
        <xsd:sequence maxOccurs="unbounded">
          <xsd:choice>
            <xsd:element ref="xforms:choices"/>
            <xsd:element ref="xforms:item"/>
            <xsd:element ref="xforms:itemset"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="value">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="item">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xforms:caption"/>
        <xsd:element ref="xforms:value"/>
        <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="itemset">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xforms:caption"/>
        <xsd:element ref="xforms:value"/>
        <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindAllAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:group name="optionalUIChildren">
```

```
  <xsd:sequence>
    <xsd:choice>
      <xsd:element ref="xforms:help"/>
      <xsd:element ref="xforms:hint"/>
      <xsd:element ref="xforms:alert"/>
      <xsd:group ref="xforms:actionGroup"/>
      <xsd:element ref="xforms:extension"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>
<xsd:element name="input">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attribute name="inputMode" type="xsd:string" use="optional"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="textarea">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attribute name="inputMode" type="xsd:string" use="optional"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="secret">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attribute name="inputMode" type="xsd:string" use="optional"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="upload">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
    <xsd:attribute name="mediaType" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
<xsd:group name="listChoices">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element ref="xforms:item"/>
      <xsd:element ref="xforms:itemset"/>
      <xsd:element ref="xforms:choices"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:group>
<xsd:element name="selectOne">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:listChoices" maxOccurs="unbounded"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
    <xsd:attribute name="selectUI" use="optional">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="radio"/>
          <xsd:enumeration value="checkbox"/>
          <xsd:enumeration value="menu"/>
          <xsd:enumeration value="listbox"/>
          <xsd:enumeration value="combo"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="selection" use="optional" default="closed">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="open"/>
          <xsd:enumeration value="closed"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
<xsd:element name="selectMany">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:listChoices" maxOccurs="unbounded"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
    <xsd:attribute name="selectUI" use="optional">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="radio"/>
          <xsd:enumeration value="checkbox"/>
          <xsd:enumeration value="menu"/>
          <xsd:enumeration value="listbox"/>
          <xsd:enumeration value="combo"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="selection" use="optional" default="closed">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="open"/>
          <xsd:enumeration value="closed"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
<xsd:element name="range">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
    <xsd:attribute name="start" type="xsd:string" use="optional"/>
    <xsd:attribute name="end" type="xsd:string" use="optional"/>
    <xsd:attribute name="stepSize" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="button">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="output">
  <xsd:complexType>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attribute name="ref" type="xsd:string" use="optional"/>
    <xsd:attribute name="model" type="xsd:string" use="optional"/>
    <xsd:attribute name="format" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="submit">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xforms:caption"/>
      <xsd:group ref="xforms:optionalUIChildren" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attribute name="submitInfo" type="xsd:IDREF" use="required"/>
  </xsd:complexType>
  </xsd:element>
  <!--
XForms Actions
-->
  <xsd:attributeGroup name="XMLEvents">
    <xsd:attribute ref="ev:event"/>
    <xsd:attribute ref="ev:observer"/>
    <xsd:attribute ref="ev:target"/>
    <xsd:attribute ref="ev:handler"/>
    <xsd:attribute ref="ev:phase"/>
    <xsd:attribute ref="ev:propagate"/>
    <xsd:attribute ref="ev:defaultAction"/>
  </xsd:attributeGroup>
  <xsd:group name="actionGroup">
    <xsd:choice>
      <xsd:element ref="xforms:action"/>
      <xsd:group ref="xforms:actions"/>
    </xsd:choice>
  </xsd:group>
  <xsd:element name="action">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:group ref="xforms:actions"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:group name="actions">
    <xsd:choice>
      <xsd:element ref="xforms:dispatch"/>
      <xsd:element ref="xforms:refresh"/>
      <xsd:element ref="xforms:revalidate"/>
      <xsd:element ref="xforms:recalculate"/>
      <xsd:element ref="xforms:setFocus"/>
      <xsd:element ref="xforms:loadURI"/>
      <xsd:element ref="xforms:setValue"/>
      <xsd:element ref="xforms:submitInstance"/>
      <xsd:element ref="xforms:resetInstance"/>
      <xsd:element ref="xforms:insert"/>
      <xsd:element ref="xforms:delete"/>
      <xsd:element ref="xforms:setRepeatCursor"/>
      <xsd:element ref="xforms:toggle"/>
      <xsd:element ref="xforms:script"/>
      <xsd:element ref="xforms:message"/>
    </xsd:choice>
  </xsd:group>
  <xsd:element name="dispatch">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="name" type="xsd:NMTOKEN" use="required"/>
      <xsd:attribute name="target" type="xsd:IDREF" use="required"/>
      <xsd:attribute name="bubbles" type="xsd:boolean" use="optional" default="true"/>
      <xsd:attribute name="cancelable" type="xsd:boolean" use="optional" default="true"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="refresh">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="recalculate">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="revalidate">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="setFocus">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="idref" type="xsd:IDREF" use="required"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="loadURI">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attribute ref="xlink:href" use="required"/>
      <xsd:attribute ref="xlink:show" use="optional"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="setValue">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attributeGroup ref="xforms:horzAttrs"/>
          <xsd:attribute name="id" type="xsd:ID" use="optional"/>
          <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
          <xsd:attribute name="value" type="xsd:string"/>
          <xsd:attributeGroup ref="xforms:XMLEvents"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="submitInstance">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="submitInfo" type="xsd:IDREF" use="optional"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="resetInstance">
```

```
  <xsd:complexType>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:XMLEvents"/>
    <xsd:attribute name="model" type="xsd:IDREF"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="insert">
  <xsd:complexType>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindAllAttributes"/>
    <xsd:attribute name="at" type="xsd:string" use="required"/>
    <xsd:attribute name="position" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="before"/>
          <xsd:enumeration value="after"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attributeGroup ref="xforms:XMLEvents"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="delete">
  <xsd:complexType>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attributeGroup ref="xforms:bindAllAttributes"/>
    <xsd:attribute name="at" type="xsd:string" use="required"/>
    <xsd:attributeGroup ref="xforms:XMLEvents"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="setRepeatCursor">
  <xsd:complexType>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attribute name="repeat" type="xsd:IDREF" use="required"/>
    <xsd:attribute name="cursor" type="xsd:string" use="required"/>
    <xsd:attributeGroup ref="xforms:XMLEvents"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="toggle">
  <xsd:complexType>
    <xsd:attributeGroup ref="xforms:horzAttrs"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attribute name="case" type="xsd:IDREF" use="required"/>
    <xsd:attributeGroup ref="xforms:XMLEvents"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="script">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attributeGroup ref="xforms:horzAttrs"/>
        <xsd:attribute name="id" type="xsd:ID" use="optional"/>
        <xsd:attribute name="type" type="xsd:string" use="required"/>
        <xsd:attribute name="role" type="xsd:string" use="optional"/>
        <xsd:attributeGroup ref="xforms:XMLEvents"/>
      </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="message">
    <xsd:complexType>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attribute name="level" type="xsd:string" use="required"/>
      <xsd:attributeGroup ref="xforms:XMLEvents"/>
    </xsd:complexType>
  </xsd:element>
  <!--
Advanced User Interface
-->
  <xsd:element name="group">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:element ref="xforms:caption" minOccurs="0"/>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="switch">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:element ref="xforms:case"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
      <xsd:attributeGroup ref="xforms:bindFirstAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
      <xsd:attribute name="default" type="xsd:IDREF" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="case">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attribute name="selected" type="xsd:boolean" use="optional"/>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="repeat">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xforms:horzAttrs"/>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xforms:bindAllAttributes"/>
      <xsd:attributeGroup ref="xforms:commonUIAttributes"/>
      <xsd:attribute name="startIndex" type="xsd:positiveInteger" use="optional"/>
      <xsd:attribute name="number" type="xsd:nonNegativeInteger" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--
New simpleTypes
-->
  <xsd:simpleType name="QNameList">
    <xsd:list itemType="xsd:QName"/>
  </xsd:simpleType>
  <xsd:simpleType name="QNameButNotNCNAME">
    <xsd:restriction base="xsd:QName">
      <xsd:pattern value="[^:]+:[^:]+"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="listItem">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="/S+"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="listItems">
    <xsd:list itemType="xforms:listItem"/>
  </xsd:simpleType>
</xsd:schema>
```

A.1 Schema for XLink

This schema is not normative with respect to XLink, although it is considered a normative part of the XForms definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.w3.org/1999/xlink"
      xmlns:xl="http://www.w3.org/1999/xlink"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!--
  This schema is in no way normative for XLink; it functions only as a part of the
  Schema for XForms to allow proper assessment of XForms documents and fragments.

  See the XForms specification for details.
-->
  <xsd:attribute name="href" type="xsd:anyURI"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="role" type="xsd:anyURI"/>
  <xsd:attribute name="arcrole" type="xsd:anyURI"/>
  <xsd:attribute name="title" type="xsd:string"/>
  <xsd:attribute name="actuate">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="onLoad"/>
        <xsd:enumeration value="onRequest"/>
        <xsd:enumeration value="other"/>
        <xsd:enumeration value="none"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="show">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="new"/>
        <xsd:enumeration value="replace"/>
        <xsd:enumeration value="embed"/>
        <xsd:enumeration value="other"/>
        <xsd:enumeration value="none"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="label" type="xsd:NCName"/>
  <xsd:attribute name="from" type="xsd:NCName"/>
  <xsd:attribute name="to" type="xsd:NCName"/>
</xsd:schema>
```

A.2 Schema for XML Events

This schema is not normative with respect to XML Events, although it is considered a normative part of the XForms definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by Micah Dubinko (XForms WG) -->
<xsd:schema targetNamespace="http://www.w3.org/2001/xml-events"
     xmlns:ev="http://www.w3.org/2001/xml-events"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="unqualified">
  <!--
  This schema is in no way normative for XML Events; it functions only as a part of the
  Schema for XForms to allow proper assessment of XForms documents and fragments.

  See the XForms specification for details.
-->
  <xsd:attribute name="event" type="xsd:NMTOKEN"/>
  <xsd:attribute name="observer" type="xsd:IDREF"/>
  <xsd:attribute name="target" type="xsd:IDREF"/>
  <xsd:attribute name="handler" type="xsd:anyURI"/>
  <xsd:attribute name="phase">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="capture"/>
        <xsd:enumeration value="default"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="propagate">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="stop"/>
        <xsd:enumeration value="continue"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="defaultAction">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="cancel"/>
        <xsd:enumeration value="perform"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:schema>
```

B References

B.1 Normative References

RFC 2388
RFC 2388: Returning Values from Forms: multipart/form-data , L. Masinter, 1998. Available at: http://www.ietf.org/rfc/rfc2388.txt.

XForms Req
XForms Requirements, Micah Dubinko, Dave Raggett, Sebastian Schnitzenbaumer, Malte Wedel, 2001. W3C Working Draft: available at: http://www.w3.org/TR/xhtml-forms-req.

XML Events
XML Events - An events syntax for XML , Steven Pemberton, T. V. Raman, Shane P. McCarron, 2001. W3C Last Call Working Draft available at: http://www.w3.org/TR/xml-events/.

XLink
XML Linking Language (XLink) Version 1.0, Steve DeRose, Eve Maler, David Orchard, 2001. W3C Recommendation available at: http://www.w3.org/TR/xlink/.

XML 1.0
Extensible Markup Language (XML) 1.0 (Second Edition), Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 2000. W3C Recommendation: available at: http://www.w3.org/TR/REC-xml

XML Names
Namespaces in XML, Tim Bray, Dave Hollander, Andrew Layman, 1999. W3C Recommendation available at: http://www.w3.org/TR/REC-xml-names.

XPath 1.0
XML Path Language (XPath) Version 1.0, James Clark, Steve DeRose, 1999. W3C Recommendation available at: http://www.w3.org/TR/xpath.

XML Schema part 1
XML Schema Part 1: Structures, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-1/.

XML Schema part 2
XML Schema Part 2: Datatypes, Paul V. Biron, Ashok Malhotra, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-2/.

XSLT
XSL Transformations (XSLT) Version 1.0, James Clark, 1999. W3C Recommendation available at: http://www.w3.org/TR/xslt.

B.2 Informative References

AUI97
Auditory User Interfaces--Toward The Speaking Computer , T. V. Raman, Kluwer Academic Publishers, 1997. ISBN:0-7923-9984-6.

CSS2
Cascading Style Sheets, level 2 (CSS2) Specification, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, 1998. W3C Recommendation available at: http://www.w3.org/TR/REC-CSS2.

DOM2 Events
Document Object Model (DOM) Level 2 Events Specification, Tom Pixley, 2000. W3C Recommendation available at: http://www.w3.org/TR/DOM-Level-2-Events/.

DDJ-ArrayDoubling
Resizable Arrays, Heaps and Hash Tables, John Boyer, Doctor Dobb's Journal, CMP Media LLC, January 1998 Issue.

FIMS
Form Interface Management System, ISO/IEC DIS 11730, 1992. available at: http://gatekeeper.research.compaq.com/pub/standards/sql/fims.txt

P3P 1.0
The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, Joseph Reagle, 2001. W3C Last Call Working Draft available at:

http://www.w3.org/TR/P3P/.

XHTML 1.0
XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0 , Steven Pemberton, et al., 2000. W3C Recommendation available at: http://www.w3.org/TR/xhtml1.

XML Schema part 0
XML Schema Part 0: Primer, David C. Fallside, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-0/.

C Recalculation Sequence Algorithm

XForms Processors are free (and encouraged) to skip or optimize any steps in this algorithm, as long as the end result is the same. The XForms recalculation algorithm considers model items and model item properties to be vertices in a directed graph. Edges between the vertices represent computational dependencies between vertices.

Following is the default handling for a `recalculate` action. Action `recalculate` is defined in **10.3 recalculate**.

1. A master dependency directed graph is created as detailed in **C.1 Details on Creating the Master Dependency Directed Graph**.
2. To provide consistent behavior, implementations must reduce the number of vertices to be processed by computing a pertinent dependency subgraph consisting only of vertices and edges that are reachable from nodes that require recomputation. This is detailed in **C.2 Details on Creating the Pertinent Dependency Subgraph**. Note that on a first recomputation (such as on form load), the pertinent dependency subgraph will be the same as the master dependency directed graph.
3. A topological sort is performed on the vertices of the pertinent dependency subgraph, resulting in an order of evaluation in which each vertex is evaluated only after those vertices on which it depends and before all vertices which depend on it. This is detailed in **C.3 Details on Computing Individual Vertices**.
4. The `recalculate` process completes.

C.1 Details on Creating the Master Dependency Directed Graph

The master dependency directed graph can be considered an array with one record for each vertex, each having the following fields:

InstanceNode: a reference to the associated instance data node
type: indicates the aspect of the instance node represented by the vertex (the text content or a model item property such as readOnly or required)
depList: a list of vertices that refer to this vertex
inDegree: the number of vertices on which this vertex depends
visited: a flag used to ensure vertices are not added to a subgraph multiple times
index: an association between vertices in the master dependency directed graph and a subgraph

The `depList` for each vertex is assigned to be the referenced XML nodes of a given instance node, which are obtained by parsing the computed expression in the node (e.g., the calculate, relevant, readOnly, or required property). Any expression violating any Binding Expression Constraint causes a fatal exception, terminating the `recalculate` process.

The `depList` for a vertex `v` is assigned to be the vertices other than `v` whose computational expressions reference `v` (described below). Vertex `v` is excluded from its own `depList` to allow self-references to occur without causing a circular reference exception.

A computational expression appearing in a `calculate` attribute controls the text content (value) of one or more instance nodes. A vertex exists for each instance node to represent the expression in the context of the node.

Likewise, computational expressions for model item properties such as `readOnly` and `required` are applied to one or more instance nodes, and vertices are created to represent such expressions in the context of each applicable node. The computational expression of each vertex must be examined to determine the XML nodes to which it refers. Any expression violating any Binding Expression Constraint causes a fatal exception, terminating the `recalculate` process. A computation expression refers to a vertex `v` if a subexpression indicates the InstanceNode for `v` and `v` represents the instance node text content (its value). In this version of XForms, model item properties such as `readOnly` and `required` cannot be referenced in an expression.

C.2 Details on Creating the Pertinent Dependency Subgraph

If all calculations must be performed, which is the case on form load, then the pertinent dependency subgraph is simply a duplicate of the master dependency directed graph. If the recalculation algorithm is invoked with a list of changed instance data nodes since the last recalculation, then the pertinent dependency subgraph is obtained by exploring the paths of edges and vertices in the computational dependency directed graph that are reachable from each vertex in the change list. The method of path exploration can be depth first search, a suitable version of which appears in the pseudo-code below.

Example: Sample Algorithm to Create the Pertinent Dependency Subgraph

This algorithm creates a pertinent dependency subgraph `S` from a list of changed instance data nodes `L<sub>c</sub>`. Variables such as `v` and `w` represent vertices in the master dependency directed graph. The same variables ending with `S` indicate vertices in the pertinent dependency subgraph `S`.

```
// Use depth-first search to explore master digraph subtrees rooted at
// each changed vertex. A 'visited' flag is used to stop exploration
// at the boundaries of previously explored subtrees (because subtrees
// can overlap in directed graphs).
for each vertex r in Lc
  if r is not visited
  {
    Push the pair (NIL, r) onto a stack
    while the stack is not empty
    {
      (v, w) = pop dependency pair from stack
      if w is not visited
      {
         Set the visited flag of w to true
         Create a vertex wS in S to represent w
         Set the index of w equal to the array location of wS
         Set the index of wS equal to the array location of w
         Set the InstanceNode of wS equal to the InstanceNode of w
         Set the type of wS equal to the type of w
         For each dependency node x of w
             Push the pair (w, x) onto the stack
      }
      else Obtain wS from index of w
      if v is not NIL
      {
         Obtain vS from index of v
         Add dependency node for wS to vS
         Increment inDegree of wS
      }
    }
  }

// Now clear the visited flags set in the loop above
```

```
for each vertex vS in S
{
    Obtain v from index of vS
    Assign false to the visited flag of v
}
```

Note that the number of vertices and dependency nodes in the pertinent dependency subgraph is not known beforehand, but a method such as array doubling (see [DDJ-ArrayDoubling]) can be used to ensure that building the subgraph is performed in time linear in the size of `S`.

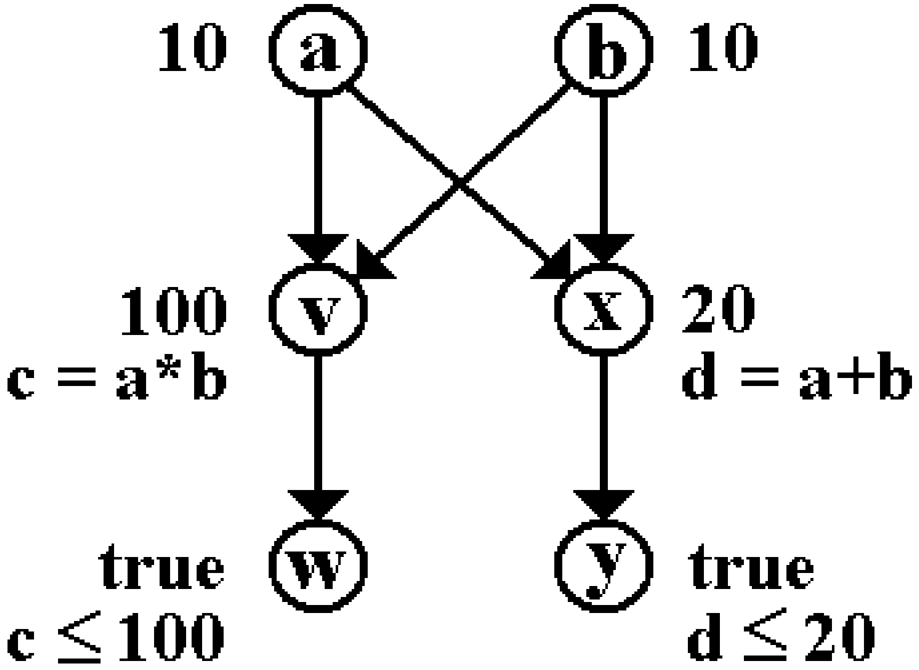
C.3 Details on Computing Individual Vertices

The following steps process vertices, resulting in a recalculated form:

1. A vertex with inDegree of 0 is selected for evaluation and removed from the pertinent dependency subgraph. In the case where more than one vertex has inDegree zero, no particular ordering is specified. If the pertinent dependency subgraph contains vertices, but none have an inDegree of 0, then the calculation structure of the form has a loop, and a fatal exception must be thrown, terminating the recalculate event.
2. If the vertex corresponds to a computed item, computed expressions are evaluated as follows:
    1. `calculate`: If the value of the model item changes, the corresponding instance data is updated and the dirty flag is set.
    2. `relevant`, `readOnly`, `required`, `isValid`: If any or all of these computed properties change, the new settings are immediately placed into effect for associated form controls.
3. For each vertex in the `depList` of the removed vertex, decrement the inDegree by 1.
4. If no vertices remain in the pertinent dependency subgraph, then the calculation has successfully completed. Otherwise, repeat this sequence from step 1.

C.4 Example of Calculation Processing

For example, consider six vertices `a`, `b`, `v`, `w`, `x`, and `y`. Let `a` and `b` represent the text content of instance nodes that will be set by a binding from user input controls. Let `v` and `w` be vertices representing the calculated value and the validity property of a third instance node `c`. These vertices would result from a `bind` element `B` with `calculate` and `isValid` attributes and a `ref` attribute that indicates `c`. Suppose that the value of `c` is the product of `a` and `b` and that the value is only valid if it does not exceed 100. Likewise, suppose `x` and `y` are vertices representing the calculated value and the validity property of a fourth instance node `d`. Let the value of `d` be the sum of `a` and `b`, and let `d` be valid if the value does not exceed 20. The figure below depicts the dependency digraph for this example.



Vertices a and b have edges leading to v and x because these vertices represent the calculate expressions of c and d, which reference a and b to compute their product and sum, respectively. Similarly, v and x have directed edges to w and y, respectively, because w and y represent the isValid expressions of c and d, which reference the values of c and d to compare them with boundary values.

If a and b are initially equal to 10, and the user changes a to 11, then it is necessary to first recalculate v (the value of c) then recalculate w (the validity property of the value of c). Likewise, x (the value of d) must be recalculated before recalculating y (the validity property of the value of d). In both cases, the validity of the value does not change to false until after the new product and sum are computed based on the change to a. However, there are no interdependencies between v and x, so the product and sum could be computed in either order.

The pertinent subgraph excludes b and only vertex a has in-degree of zero. The vertex a is processed first. It is not a computed vertex, so no recalculation occurs on a, but its removal causes v and x to have in-degree zero. Vertex v is processed second. Its value changes to 121, and its removal drops the in-degree of vertex w to zero. Vertex x is processed next, changing value to 21. When x is removed, its neighbor y drops to in-degree zero. The fourth and fifth iterations of this process recalculate the validity of w and y, both of which change to false.

D Input Modes

The attribute inputMode provides a *hint* to the user agent to select an appropriate input mode for the text input expected in an associated form control. The input mode may be a keyboard configuration, an input method editor (also called front end processor) or any other setting affecting input on the device(s) used.

Using inputMode, the author can give hints to the agent that make form input easier for the user. Authors should provide inputMode attributes wherever possible, making sure that the values used cover a wide range of devices.

D.1 inputMode Attribute Value Syntax

The value of the `inputMode` attribute is a white space separated list of tokens. Tokens are either sequences of alphabetic letters or absolute URIs. The later can be distinguished from the former by noting that absolute URIs contain a ':'. Tokens are case-sensitive. All the tokens consisting of alphabetic letters only are defined in this specification, in **D.3 List of Tokens** (or a successor of this specification).

This specification does not define any URIs for use as tokens, but allows others to define such URIs for extensibility. This may become necessary for devices with input modes that cannot be covered by the tokens provided here. The URI should dereference to a human-readable description of the input mode associated with the use of the URI as a token. This description should describe the input mode indicated by this token, and whether and how this token modifies other tokens or is modified by other tokens.

D.2 User Agent Behavior

Upon entering an empty form control with an `inputMode` attribute, the user agent should select the input mode indicated by the `inputMode` attribute value. User agents should not use the `inputMode` attribute to set the input mode when entering a form control with text already present. To set the appropriate input mode when entering a form control that already contains text, user agents should rely on platform-specific conventions.

User agents should make available all the input modes which are supported by the (operating) system/device(s) they run on/have access to, and which are installed for regular use by the user. This is typically only a small subset of the input modes that can be described with the tokens defined here.

The following simple algorithm is used to define how user agents match the values of an `inputMode` attribute to the input modes they can provide. This algorithm does not have to be implemented directly; user agents just have to behave as if they used it. The algorithm is not designed to produce "obvious" or "desirable" results for every possible combination of tokens, but to produce correct behavior for frequent token combinations and predictable behavior in all cases.

First, each of the input modes available is represented by one or more lists of tokens. An input mode may correspond to more than one list of tokens; as an example, on a system set up for a Greek user, both "greek upper" and "user upper" would correspond to the same input mode. No two lists will be the same.

Second, the `inputMode` attribute is scanned from front to back. For each token `t` in the `inputMode` attribute, if in the remaining list of tokens representing available input modes there is any list of tokens that contains `t`, then all lists of tokens representing available input modes that do not contain `t` are removed. If there is no remaining list of tokens that contains `t`, then `t` is ignored.

Third, if one or more lists of tokens are left, and they all correspond to the same input mode, then this input mode is chosen. If no list is left (meaning that there was none at the start) or if the remaining lists correspond to more than one input mode, then no input mode is chosen.

Example: Assume the list of lists of tokens representing the available input modes is: {"cyrillic upper", "cyrillic lower", "cyrillic", "latin", "user upper", "user lower"}, then the following `inputMode` values select the following input modes: "cyrillic title" selects "cyrillic", "cyrillic lower" selects "cyrillic lower", "lower cyrillic" selects "cyrillic lower", "latin upper" selects "latin", but "upper latin" does select "cyrillic upper" or "user upper" if they correspond to the same input mode, and does not select any input mode if "cyrillic upper" and "user upper" do not correspond to the same input mode.

D.3 List of Tokens

Tokens defined in this specification are separated into two categories: *Script tokens* and *modifiers*. In `inputMode` attributes, script tokens should always be listed before modifiers.

D.3.1 Script Tokens

Script tokens provide a general indication the set of characters that is covered by an input mode. In most cases, script tokens correspond directly to Unicode Scripts (see http://www.unicode.org/Public/UNIDATA/Scripts.txt). Some tokens correspond to the block names in Java class java.lang.Character.UnicodeBlock (see http://java.sun.com/j2se/1.4/docs/api/java/lang/Character.UnicodeBlock.html; see also Unicode Block names at http://www.unicode.org/Public/UNIDATA/Blocks.txt). However, this neither means that an input mode has to allow input for all the characters in the script or block, nor that an input mode is limited to only characters from that specific script. As an example, a "latin" keyboard doesn't cover all the characters in the Latin script, and includes punctuation which is not assigned to the Latin script. The version of the Unicode Standards that these script names are taken from is 3.2.

D.3.2 Modifier Tokens

Modifier tokens can be added to the scripts they apply to more closely specify the kind of characters expected in the form field. Traditional PC keyboards do not need most modifier tokens (indeed, users on such devices would be quite confused if the software decided to change case on its own; CAPS lock for upperCase may be an exception). However, modifier tokens can be very helpful to set input modes for small devices.

D.4 Relationship to XML Schema pattern facets

User agents may use information available in an XML Schema pattern facet to set the input mode. Note that a pattern facet is a hard restriction on the lexical value of an instance data node, and can specify different restrictions for different parts of the data item. Attribute `inputMode` is a soft hint about the kinds of characters that the user may most probably start to input into the form control. Attribute `inputMode` is provided in addition to pattern facets for the following reasons:

1. The set of allowable characters specified in a pattern may be so wide that it is not possible to deduce a reasonable input mode setting. Nevertheless, there frequently is a kind of characters that will be input by the user with high probability. In such a case, `inputMode` allows to set the input mode for the user's convenience.
2. In some cases, it would be possible to derive the input mode setting from the pattern because the set of characters allowed in the pattern closely corresponds to a set of characters covered by an `inputMode` attribute value. However, such a derivation would require a lot of data and calculations on the user agent.
3. Small devices may leave the checking of patterns to the server, but will easily be able to switch to those input modes that they support. Being able to make data entry for the user easier is of particular importance on small devices.

D.5 Examples

This is an example of a form for Japanese address input. It is shown in table form; it will be replaced by actual syntax in a later version of this specification.

E Complete XForms Examples

This section presents complete XForms examples.

E.1 XForms In XHTML

```
<!--$Id: index-all.fo,v 1.4 2002/01/16 23:32:13 tvraman Exp $-->
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xforms="http://www.w3.org/2002/01/xforms"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:my="http://commerce.example.com/payment"
      xml:lang="en">
  <head>
    <title>XForms in XHTML</title>

    <xforms:model>
      <xforms:instance>
        <payment as="credit" xmlns="http://commerce.example.com/payment">
          <cc/>
          <exp/>
        </payment>
      </xforms:instance>
      <xforms:schema xlink:href="payschema.xsd"/>
      <xforms:submitInfo action="http://example.com/submit" method="post" id="s00"/>
      <xforms:bind ref="my:payment/my:cc"
                   relevant="../my:payment/@as = 'credit'"
                   required="true" type="my:cc"/>
      <xforms:bind ref="my:payment/my:exp"
                   relevant="../my:payment/@as = 'credit'"
                   required="true" type="xsd:gYearMonth"/>
    </xforms:model>
  </head>
  <body>
    ...
    <group xmlns="http://www.w3.org/2002/01/xforms" ref="my:payment">
      <selectOne ref="@as">
        <caption>Select Payment Method</caption>
        <choices>
          <item>
            <caption>Cash</caption>
            <value>cash</value>
          </item>
          <item>
            <caption>Credit</caption>
            <value>credit</value>
          </item>
        </choices>
      </selectOne>

      <input ref="my:cc">
        <caption>Credit Card Number</caption>
      </input>

      <input ref="my:exp">
        <caption>Expiration Date</caption>
      </input>

      <submit submitInfo="s00">
        <caption>Submit Form</caption>
      </submit>
    </group>
    ...
  </body>
```

```
</html>
```

E.2 Editing Hierarchical Bookmarks Using XForms

```
<?xml version="1.0"?>
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xforms="http://www.w3.org/2002/01/xforms"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:my="http://commerce.example.com/payment"
  xmlns:ev="http://www.w3.org/2001/xml-events" xml:lang="en">
  <head>
    <style type="text/css">
xforms:input.editField {
      font-weight:bold;font-size:20px;width:500px}
xforms:caption.sectionCaption {
      font-weight:bold;color:white;background-color:blue}
xforms:submit {font-family: Arial; font-size: 20px; font-style: bold; color: red;}
    </style>
    <title>Editing Hierarchical Bookmarks Using In An XML Browser</title>
    <xforms:model id="bookmarks">
<!--The bookmarks instance tree is shown inline for
      the sake of this example.
XML browser XSmiles would use
 <xforms:instance xlink:href="bookmarks.xml"/>.
-->
      <xforms:instance xmlns="">
        <bookmarks>
          <section name="main">
            <bookmark href="http://www.xsmiles.org/demo/demos.xml" name="Main page"/>
          </section>
          <section name="demos">
            <bookmark href="http://www.xsmiles.org/demo/fo/images.fo" name="images"/>
            <bookmark href="http://www.xsmiles.org/demo/fo/xforms-ecma.xml" name="xforms-e
            <bookmark href="http://www.xsmiles.org/demo/fo/sip.fo" name="sip"/>
          </section>
          <section name="misc">
            <bookmark href="sip:mhonkala@xdemo.tml.hut.fi" name="call: mhonkala"/>
            <bookmark href="sip:tvraman@examples.com" name="call: tvraman"/>
            <bookmark href="http://www.xsmiles.org/demo/links.xml" name="Links"/>
          </section>
          <section name="XForms">
            <bookmark href="file:/C:/source/xsmiles/demo/xforms/xforms-xmlevents.xml" name
            <bookmark href="file:/C:/source/xsmiles/demo/xforms/model3.xml" name="model3"/
            <bookmark href="file:/C:/source/xsmiles/demo/xforms/repeat.fo" name="repeat +
          </section>
        </bookmarks>
      </xforms:instance>
      <xforms:submitInfo id="s01" method="post" action="http://www.examples.com/"/>
    </xforms:model>
  </head>
  <body>
    <xforms:repeat nodeset="bookmarks/section" id="repeatSections">
      <xforms:input ref="@name" class="editField">
        <xforms:caption class="sectionCaption">Section</xforms:caption>
      </xforms:input>
```

```
<!-- BOOKMARK REPEAT START -->
      <xforms:repeat nodeset="bookmark" id="repeatBookmarks">
        <xforms:input ref="@name">
          <xforms:caption>Bookmark name</xforms:caption>
        </xforms:input>
        <xforms:input ref="@href">
          <xforms:caption>URL</xforms:caption>
        </xforms:input>
      </xforms:repeat>
    </xforms:repeat>
    <p>
<!-- INSERT BOOKMARK BUTTON -->
      <xforms:button id="insertbutton">
        <xforms:caption>Insert bookmark</xforms:caption>
        <xforms:insert nodeset="/bookmarks/section[xforms:cursor('repeatSections')]/bookma
      </xforms:button>
<!-- DELETE BOOKMARK BUTTON -->
      <xforms:button id="delete">
        <xforms:caption>Delete bookmark</xforms:caption>
        <xforms:delete nodeset="/bookmarks/section[xforms:cursor('repeatSections')]/bookma
      </xforms:button>
    </p>
    <p>
<!-- INSERT SECTION BUTTON -->
      <xforms:button id="insertsectionbutton">
        <xforms:caption>Insert section</xforms:caption>
        <xforms:insert nodeset="/bookmarks/section" at="xforms:cursor('repeatSections')" p
      </xforms:button>
<!-- DELETE SECTION BUTTON -->
      <xforms:button id="deletesectionbutton">
        <xforms:caption>Delete section</xforms:caption>
        <xforms:delete nodeset="/bookmarks/section" at="xforms:cursor('repeatSections')" e
      </xforms:button>
    </p>
<!-- SUBMIT BUTTON -->
    <xforms:submit submitInfo="s01">
      <xforms:caption>Save</xforms:caption>
      <xforms:hint>Click to submit</xforms:hint>
    </xforms:submit>
  </body>
</html>
```

F Changelog (Non-Normative)

This section enumerates substantive changes since the last public version of the XForms 1.0 specification. See the diff-marked version for detailed diff-marks.

- The XForms namespace in this version is http://www.w3.org/2002/01/xforms.
- Rearranged chapter and section order.
- Added clarification on nested repeats; disallowed switch inside repeat.
- Added appendix containing complete XForms examples (**E Complete XForms Examples**).
- Clarified that element instance can contain content in any namespace, including the XForms namespace.
- Clarified that serialized instance data is wrapped in element instanceData if required for single-rootedness.
- Clarified that the IDL function getInstanceData() must return a singly rooted document.

- New material on extension functions and interoperability.
- Clarified the behavior of XForms Action toggle.
- Clarified positioning of cursor after insert/delete operation, and delete from empty.

Additionally, the following changes were made to the Schema for XForms:

```
18-Dec TVR Require attribute submitinfo on element <submit>
18-Dec TVR Add attribute role on element <script>
28-Dec MJD Change content model of <bind> from empty to bind*
02-Jan MJD Allow <itemset> inside <choices>
02-Jan MJD Cleanup: no explicit numberOrUnbounded simpleType
02-Jan MJD Cleanup: removed selectUIType simpleType
02-Jan MJD Cleanup: schema now validates attribute mediaTypeExtension on element <submitIn
02-Jan MJD Cleanup: schema now validates attribute method on element <submitInfo>
02-Jan MJD Cleanup: schema now validates attribute replace on element <submitInfo>
08-Jan MJD Allowed XForms Actions as children of <submitInfo>
08-Jan MJD Added element extensionFunctions to <model>
09-Jan MJD Typo: Added missing attribute model on <resetInstance>
09-Jan MJD Typo: Fixed content model of <alert> to match <help>&<hint>
```

G Acknowledgments (Non-Normative)

This document was produced with the participation of the XForms Working Group:

Steven Pemberton, CWI (*Chair*)
Sebastian Schnitzenbaumer, Mozquito Technologies (*Chair*)
Micah Dubinko, Cardiff (*Editor*)
Peter Stark, Ericsson
Mikko Honkala, Helsinki University Of Technology
Roland Merrick, IBM (*Editor*)
T. V. Raman, IBM (*Editor*)
Linda Bucsay Welsh, Intel (*Until April 2001*)
Gavin McKenzie, JetForm Corporation (*Until January 2001*)
Rob McDougall, JetForm Corporation (*Until January 2001*)
John McCarthy, Lawrence Berkeley National Laboratory (*Until November 2000*)
Frank Olken, Lawrence Berkeley National Laboratory (*Until November 2000*)
Ray Waldin, Lexica, LLC
Tantek Çelik, Microsoft
Panagiotis Reveliotis, Phillips (*Until December 2000*)
David Cleary, Progress Software
John Boyer, PureEdge Solutions Inc
Mike Mansell, PureEdge Solutions Inc (*Until March 2001*)
Josef Dietl, Mozquito Technologies
Doug Dominiak, Openwave
Michael Fergusson, Softquad (*Until January 2001*)
Dave Raggett, W3C/Openwave (*W3C Staff Contact until December 2000*)
Leigh Klotz, Xerox
Frank Boumphrey, HTML Writer's Guild (*Until November 2000*)
Dave Navarro, WebGeek Inc.
Dave Hyatt, Netscape/AOL
Eric Pollmann, Netscape/AOL
Tom Butcher, OpenDesign
K. P. Lee, Phillips
Roli Wendorf, Phillips
Ted Wugofski, Openwave

- Josef Dietl, Mozquito Technologies
- Zoe Lacroix, SurroMed, Inc.
- Masayasu Ishikawa, W3C (*Staff Contact until September 2001*)
- Thierry Michel, W3C (*W3C staff Contact*)

The XForms Working Group has benefited in its work from the participation of Invited Experts:

- Tom Schnetlage, University of Berkeley
- Dan Gillman, Federal Bureau of Labor Statistics
- Eliot Christian, U.S. Geological Survey

Note:

Editor Acknowledgments: Previous versions of this document were edited with assistance from Dave Raggett (until December 2000) and Linda Bucsay Welsh (until April 2001). Martin Dürst edited the section on input modes.

Note:

Additional Acknowledgments: The editors would like to thank Kai Scheppe, Malte Wedel and Götz Bock for constructive criticism on early versions of the binding discussion and their contributions to its present content. We thank John Boyer for authoring the sections on the recalculation sequence algorithm—see **C Recalculation Sequence Algorithm**. Finally, we would like to thank members of the public WWW-Forms@w3c.org mailing list for their careful reading of draft versions of this specification and providing constructive suggestions and criticisms.

Note:

Additional Acknowledgments: The Working Group would like to thank the following members of the XML Schema-XForms joint task force: Daniel Austin (chair), David Cleary, Micah Dubinko, Martin Dürst, David Ezell, Leigh Klotz, Noah Mendelsohn, Roland Merrick, and Peter Stark for their assistance in identifying a subset of XML Schema for use in XForms.

H Production Notes (Non-Normative)

This document was encoded in the XMLspec DTD (which has documentation available). The XML sources were transformed using xmlspec.xsl style sheet. The primary tools used for editing were SoftQuad XMetaL and EMACS with psgml and XAE. The XML was validated using XMLLint (part of the GNOME libxml package) and transformed using XSLTProc—part of the GNOME libxsl package). The multi-file HTML version was produced using the Xalan processor. The HTML versions were also produced at times with the Saxon engine. The editors used the W3C CVS repository and the W3C IRC server for collaborative authoring.