W3C Recommendation

# XForms 1.0

## W3C Recommendation 14 October 2003

**This version:**
http://www.w3.org/TR/2003/REC-xforms-20031014/
**Latest version:**
http://www.w3.org/TR/xforms/
**Previous version:**
http://www.w3.org/TR/2003/PR-xforms-20030801/

**Editors:**
Micah Dubinko, Cardiff Software <mdubinko@Cardiff.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>

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative formats: single HTML file, diff-marked HTML, Zip archive.

The English version of this specification is the only normative version. Non-normative translations may also be available.

## Abstract

XForms is an XML application that represents the next generation of forms for the Web. By splitting traditional XHTML forms into three parts—XForms 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 or SVG.

## Status of this Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.*

This document is a Recommendation of the W3C. This document has been produced by the W3C XForms Working Group as part of the XForms Activity within the W3C Interaction Domain. The authors of this document are the XForms Working Group participants.

It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The XForms Basic Profile which appeared in the XForms 1.0 Candidate Recommendation is now a standalone specification. Please refer to the XForms Basic Profile [XForms Basic] for XForms processing tailored to the needs of constrained devices and environments.

Comments on this document are welcome. Please send them to the public mailing list www-forms-editor@w3.org. (archive). It is inappropriate to send discussion email to this address.

The W3C XForms Working Group has released a public test suite for XForms 1.0 along with an implementation report. A list of current known XForms Implementations is available.

Patent disclosures relevant to this specification may be found on the XForms Working Group's public patent disclosure page, in conformance with W3C policy.

## Table of Contents

# 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 1.0 is the response to this demand, and provides a new platform-independent markup language for online interaction between a person (through an XForms Processor) and another, usually remote, 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 (http://www.w3.org/2002/xforms) **3.1 The XForms Namespace**
**html:** The XHTML namespace (http://www.w3.org/1999/xhtml) [XHTML 1.0]
**xsd:** The XML Schema namespace (http://www.w3.org/2001/XMLSchema)[XML Schema part 1]
**xsi:** The XML Schema for instances namespace (http://www.w3.org/2001/XMLSchema-instance)[XML Schema part 1]
**ev:** The XML Events namespace (http://www.w3.org/2001/xml-events)[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 **13 Glossary Of Terms**]. Links to **term**s may be specially highlighted where necessary.

The XML representations of various elements within XForms are presented using the syntax for Abstract Modules in XHTML Modularization [XHTML Modularization].

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.

# 2 Introduction to XForms

XForms has been designed on the basis of several years' experience with HTML forms. HTML Forms have formed the backbone of the e-commerce revolution, and having shown their worth, have also indicated numerous ways they could be improved.

The primary difference when comparing XForms with HTML Forms, apart from XForms being in XML, is the separation of the data being collected from the markup of the controls collecting the individual values. By doing this, it not only makes XForms more tractable by making it clear what is being submitted where, it also eases reuse of forms, since the underlying essential part of a Form is no longer irretrievably bound to the page it is used in.

A second major difference is that XForms, while designed to be integrated into XHTML, is no longer restricted only to be a part of that language, but may be integrated into any suitable markup language.

XForms has striven to improve authoring, reuse, internationalization, accessibility, usability, and device independence. 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. This speeds up form filling since it reduces the need for round trips to the server for validation.

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.

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 to 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.

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.

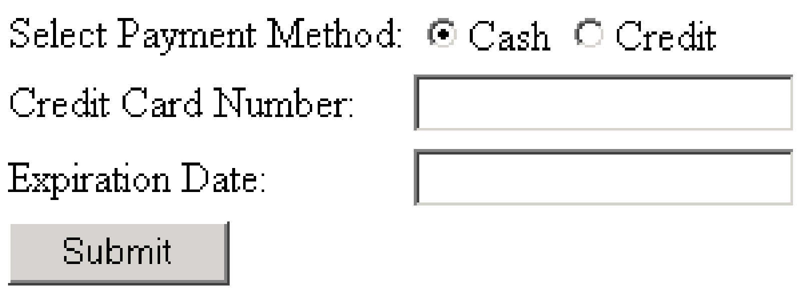
Less use of scripting

By defining XML-based declarative event handlers that cover common use cases, the majority of XForms documents can be statically analyzed, reducing the need for imperative scripts for event handlers.

## 2.1 An Example

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.

Consider a simple electronic commerce form that might be rendered as follows:



| | |
|---|---|
| Select Payment Method: | [✓ Cash] Credit |
| Credit Card Number: | |
| Expiration Date: | |

It is clear that we are collecting a value that represents whether cash or credit card is being used, and if a credit card, its number and expiration date.

This can be represented in the XForms `model` element, which in XHTML would typically be contained within the `head` section:

```
<xforms:model>
  <xforms:instance>
    <ecommerce xmlns="">
      <method/>
      <number/>
      <expiry/>
    </ecommerce>
  </xforms:instance>
  <xforms:submission action="http://example.com/submit" method="post" id="submit" inclu
</xforms:model>
```

This simply says that we are collecting three pieces of information (note that we have as yet not said anything about their types), and that they will be submitted using the URL in the `action` attribute.

XForms 1.0 defines a device-neutral, platform-independent set of form controls suitable for general-purpose use. The controls are *bound* to the XForms Model via the XForms binding mechanism, in this simple case using the `ref` attribute on the controls. In XHTML, this markup would typically appear within the `body` section (note that we have intentionally defaulted the XForms namespace prefix here):

```
<select1 ref="method">
  <label>Select Payment Method:</label>
  <item>
    <label>Cash</label>
    <value>cash</value>
  </item>
  <item>
    <label>Credit</label>
    <value>cc</value>
  </item>
</select1>
<input ref="number">
  <label>Credit Card Number:</label>
</input>
<input ref="expiry">
  <label>Expiration Date:</label>
</input>
<submit submission="submit">
  <label>Submit</label>
</submit>
```

Notice the following features of this design:

- The user interface is not hard-coded to use radio buttons. Different devices (such as voice browsers) can render the concept of "select one" as appropriate.
- Form controls always have labels 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 **2.4 Multiple Forms per Document** for details on how to author multiple forms per document)
- Markup for specifying form controls has been simplified in comparison with HTML forms.

The fact that you can bind form controls to the model like this simplifies integrating XForms into other host languages, since any form control markup may be used to bind to the model.

## 2.2 Providing XML Instance Data

The XForms Processor can directly submit the data collected as XML. In the example, the submitted data would look like this:

Example: Submitted Data

```
<ecommerce>
  <method>cc</method>
  <number>1235467789012345</number>
  <expiry>2001-08</expiry>
</ecommerce>
```

XForms processing keeps track of the state of the partially filled form through this instance data. Initial values for the instance data may be provided or left empty as in the example. Element `instance` essentially holds a skeleton XML document that gets updated as the user fills out the form. It 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. Here is an alternative version of the earlier example:

Example: Model

```
<xforms:model>
  <xforms:instance>
    <payment method="cc" xmlns="http://commerce.example.com/payment">
      <number/>
      <expiry/>
    </payment>
  </xforms:instance>
  <xforms:submission action="http://example.com/submit" method="post" includenamespacep
</xforms:model>
```

In this case the submitted data would look like this:

Example: Submitted Data

```
<payment method="cc" xmlns="http://commerce.example.com/payment">
  <number>1235467789012345</number>
  <expiry>2001-08</expiry>
</payment>
```

This design has features worth calling out:

- There is complete flexibility in the structure of the XML instance data, including the use of attributes. Notice that XML namespaces are used, and that a wrapper element of the author's choosing contains the instance data.
- Empty elements `number` and `expiry` serve as place-holders in the XML structure, and will be filled in with form data provided by the user.
- An initial value (`"cc"`) for the form control is provided through the instance data, in this case an attribute `method`. In the submitted XML, this initial value will be replaced by the user input, if the user changes the form control displaying that data.

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

Example: Binding Form Controls to Instance Nodes with `ref`

```
... xmlns:my="http://commerce.example.com/payment"
  ...
  <xforms:select1 ref="@method">...</xforms:select1>
  ...
  <xforms:input ref="my:number">...</xforms:input>
  ...
  <xforms:input ref="/my:payment/my:expiry">...</xforms:input>
```

Binding expressions are based on XPath [XPath 1.0], including the use of the `@` character to refer to attributes, as seen here. Note that for illustrative purposes, the first two expressions make use of the XPath context node, which defaults to the top-level element (here `my:payment`). The third expression shows an absolute path.

## 2.3 Constraining Values

XForms allows data to be checked for validity as the form is being filled. In the absence of specific information about the types of values being collected, all values are returned as strings, but it is possible to assign types to values in the instance data. In this example, `number` should accept digits only, and should have between 14 and 18 digits and `expiry` should accept only valid month/date combinations.

Furthermore, the credit card information form controls for `number` and `expiry` are only relevant if the `"cc"` option is chosen for `method`, but are required in that case.

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

Example: Declarative Validation with Model Item Properties

```
... xmlns:my="http://commerce.example.com/payment"...

  <xforms:model>
    ...
    <xforms:bind nodeset="/my:payment/my:number"
                 relevant="/my:payment/@method = 'cc'"
                 required="true()"
                 type="my:ccnumber"/>
    <xforms:bind nodeset="/my:payment/my:expiry"
                 relevant="/my:payment/@method = 'cc'"
                 required="true()"
                 type="xsd:gYearMonth"/>
    <xsd:schema ...>
      ...
      <xsd:simpleType name="ccnumber">
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="\d{14,18}"/>
        </xsd:restriction>
      </xsd:simpleType>
      ...
    </xsd:schema>
  </xforms:model>
```

**Note:**

In the above example, the `relevant` expression uses absolute XPath notation (beginning with `/`) because the evaluation context nodes for computed expressions are determined by the `bind` `ref` binding expression (see **7.4 Evaluation Context**), and so any relative node path in the first `bind` `relevant` above would be relative to `/my:payment/my:number`

## 2.4 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, each with an `id` attribute so that they can be referenced from elsewhere in the containing document.

In addition, form controls should specify which `model` element contains the instance data to which they bind. This is accomplished through a `model` attribute that is part of the binding attributes. If no `model` attribute is specified on the binding element, the nearest ancestor binding element's `model` attribute is used, and failing that, the first XForms Model in document order is used. This technique is called 'scoped resolution', and is used frequently in XForms.

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

Example: Adding a `poll` model

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

<xforms:model id="poll">
  <xforms:instance>
    <helpful/>
  </xforms:instance>
  <xforms:submission id="pollsubmit" .../>
</xforms:model>
```

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

Example: Form Controls for `poll` model

```
<xforms:select1 ref="/helpful" model="poll">
  <xforms:label>How useful is this page to you?</xforms:label>
  <xforms:item>
    <xforms:label>Not at all helpful</xforms:label>
    <xforms:value>0</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Barely helpful</xforms:label>
    <xforms:value>1</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Somewhat helpful</xforms:label>
    <xforms:value>2</xforms:value>
  </xforms:item>
  <xforms:item>
    <xforms:label>Very helpful</xforms:label>
    <xforms:value>3</xforms:value>
  </xforms:item>
</xforms:select1>

<xforms:submit submission="pollsubmit">
  <xforms:label>Submit</xforms:label>
</xforms:submit>
```

The main difference here is the use of `model="poll"`, which identifies the instance. Note that `submit` refers to the `submission` element by ID and does not require binding attributes.

More XForms examples can be found in **G Complete XForms Examples**.

# 3 Document Structure

XForms 1.0 is an application of XML [XML 1.0] and has been designed for use within other XML vocabularies—in particular within a future version of XHTML [XHTML 1.0]. XForms always requires such a host language. This chapter discusses the structure of XForms that allow XForms to be used with other document types.

## 3.1 The XForms Namespace

The XForms namespace has the URI: `http://www.w3.org/2002/xforms`.

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

## 3.2 XForms Core Attribute Collections

### 3.2.1 Common Attributes

The Common Attribute Collection applies to every element in the XForms namespace.

anyAttribute

Foreign attributes are allowed on all XForms elements.

A host language must permit an attribute of type `xsd:ID` on each XForms element.

### 3.2.2 Linking Attributes

The Linking Attributes Collection applies to XForms elements which include a link to a remote resource.

src

The `src` attribute assigns a URI to be automatically retrieved.

**Note:**

Since linking attribute URIs are defined in terms of the XML Schema datatype `xsd:anyURI`, the same internationalization benefits and white space cautions apply as discussed in [XML Schema part 2].

Behavior of relative URIs in links is determined by the host language, although [XML Base] processing is strongly recommended.

**Note:**

The XForms Working Group is tracking with the HTML Working Group on a method of describing link structures.

### 3.2.3 Single-Node Binding Attributes

The following attributes define a binding between a form control or an action and an instance data node defined by an XPath expression.

ref

Binding expression interpreted as XPath. This attribute has no meaning when a `bind` attribute is present.

model

XForms Model selector. Specifies the `ID` of an XForms Model to be associated with this binding element. This attribute has no meaning for the current binding element when a `bind` attribute is present. Rules for determining the context XForms Model are located at **7.4 Evaluation Context**.

bind

Reference to a `bind` element.

One of `ref` or `bind` is required. When `bind` is used, the node is determined by the referenced `bind`.

It is an exception (**4.5.1 The xforms-binding-exception Event**) if the XForms Processor encounters a `model` `IDREF` value that refers to an `ID` not on a `model` element, or a `bind` `IDREF` value that refers to an `ID` not on a `bind` element.

**First-node rule**: When a Single-Node Binding attribute selects a node-set of size > 1, the first node in the node-set, based on document order, is used.

### 3.2.4 Node-Set Binding Attributes

The following attributes define a binding between a form control or an action and a node-set defined by the XPath expression.

nodeset

Binding expression interpreted as XPath. This attribute has no meaning when a `bind` attribute is present.

model

XForms Model selector. Specifies the `ID` of an XForms Model to be associated with this binding element. This attribute has no meaning for the current binding element when a `bind` attribute is present. Rules for determining the context XForms Model are located at **7.4 Evaluation Context**.

bind

Reference to a `bind` element.

One of `nodeset` or `bind` is required. When `bind` is used, the node-set is determined by the referenced `bind`.

It is an exception (**4.5.1 The xforms-binding-exception Event**) if the XForms Processor encounters a `model` IDREF value that refers to an id not on a `model` element, or a `bind` IDREF value that refers to an `id` not on a `bind` element.

### 3.2.5 Model Item Property Attributes

This collection contains one attribute for each model item property, with an attribute name exactly matching the name of the model item property, as defined in **6.1 Model Item Property Definitions**.

## 3.3 The XForms Core Module

The XForms Core Module defines the major structural elements of XForms, intended for inclusion in a containing document. The elements and attributes included in this module are:

| Element | Attributes | Minimal Content Model |
|---|---|---|
| model | Common, Events, functions (QNameList), schema (list of xsd:anyURI) | (instance\|xsd:schema\|submission\|bind\|Action)* |
| instance | Common, Linking | (ANY) |
| submission | Common, ref (binding-expression), bind (xsd:IDREF), action (xsd:anyURI), method ("post"\|"get"\|"put"\|"form-data-post"\|"urlencoded-post"\|qname-but-not-ncname), version (xsd:NMTOKEN), indent (xsd:boolean), mediatype (xsd:string), encoding (xsd:string), omit-xml-declaration (xsd:boolean), standalone (xsd:boolean), cdata-section-elements (QNameList), replace ("all"\|"instance"\|"none"\|qname-but-not-ncname), separator (';' \| '&'), includenamespaceprefixes (xsd:NMTOKENS) | Action* |
| bind | Common, Model Item Properties, nodeset (model-binding-expression) | (bind)* |

Elements defined in the XForms Actions module, when that module is included, are also allowed in the content model of `model` and `submission`, as shown above.

Within the containing document, these structural elements are typically not rendered.

The XForms Processor must ignore any foreign-namespaced attributes that are unrecognized, and must process unrecognized foreign-namespaced elements according to the **3.4 The XForms MustUnderstand Module** rules.

Note that the presence of foreign namespaced elements is subject to the definition of the containing document profile.

### 3.3.1 The model Element

This element represents a form definition and is used as a container for elements that define the XForms Model. No restriction is placed on how many `model` elements may exist within a containing document.

Common Attributes: Common, Events

Attributes from XML Events are allowed on this element to facilitate creating observers. This element is not an XForms Action, and has no predefined behavior event-based behavior.

Special Attributes:

functions

> Optional space-separated list of XPath extension functions (represented by QNames) required by this XForms Model. Guidance on the use of this attribute is at **7.12 Extension Functions**.

schema

> Optional list of `xsd:anyURI` links to XML Schema documents outside this `model` element. The XForms Processor must process all Schemas listed in this attribute. Within each XForms Model, there is a limit of one Schema per namespace declaration, including inline and linked Schemas.
>
> > **Note:**
> >
> > The `schema` list may include URI fragments referring to elements located elsewhere in the containing document; e.g. `"#myschema"`.

This example shows a simple usage of `model`, with the XForms namespace defaulted:

Example: Model

```
<model id="Person" schema="MySchema.xsd">
  <instance src="http://example.com/cgi-bin/get-instance" />
  ...
</model>
```

### 3.3.2 The instance Element

This optional element contains or references initial instance data.

Common Attributes: Common

Special Attributes:

Linking Attributes

> Optional link to externally defined initial instance data. If the link traversal fails, it is treated as an exception (**4.5.2 The xforms-link-exception Event**).

If both an attribute and inline content are provided, the linked version takes precedence as described at **4.2.1 The xforms-model-construct Event**.

If the initial instance data is given by a link, then the instance data is formed by creating an XPath data model of the linked resource.

If the initial instance data is given by inline content, then instance data is obtained by first creating a detached copy of the inline content (including namespaces inherited from the enveloping ancestors), then creating an XPath data model over the detached copy. The detached copy must consist of content that would be well-formed XML if it existed in a separate document. Note that this restricts the element content of `instance` to a single child element.

> **Note:**
>
> XForms authors who need additional control over the serialization of namespace nodes can use the `includenamespaceprefixes` attribute on the `submission` element.

### 3.3.3 The submission Element

This element represents declarative instructions on what to submit, and how. Details of submit processing are described at **11 Submit**.

Common Attributes: Common

Special Attributes:

bind

> Optional reference to a `bind` element. When present, the binding reference on this attribute is used in preference to any binding reference from the `ref` attribute.

ref

> Optional selector binding expression enabling submission of a portion of the instance data. The selected node, and all descendants, are selected for submission. The default value is "/".

action

> Required destination URI for submitting instance data.

method

> Required attribute specifying the protocol to be used to transmit the serialized instance data. There is no default value.

version

> Optional attribute specifying the `version` of XML to be serialized.

indent

> Optional attribute specifying whether the serializer should add extra white space nodes for readability.

mediatype

Optional attribute specifying the mediatype for XML instance serialization. Authors should ensure that the type specified is compatible with `application/xml`.

encoding

Optional attribute specifying an encoding for serialization.

omit-xml-declaration

Optional attribute specifying whether to omit the XML declaration on the serialized instance data.

standalone

Optional attribute specifying whether to include a standalone declaration in the serialized XML.

cdata-section-elements

Optional attribute specifying element names to be serialized with CDATAsections.

replace

Optional attribute specifying how the information returned after submit should be applied. In the absence of this attribute, "all" is assumed.

separator

Optional attribute specifying the separator character between name/value pairs in urlencoding. The default value is ';'.

includenamespaceprefixes

Optional attribute providing control over namespace serialization. If absent, all namespace nodes present in the instance data are considered for serialization. If present, specifies list of namespace prefixes to consider for serialization, in addition to those visibly utilized. As in [Exc-C14N], the special value `#default` specifies the default namespace.

The following examples show how various options on element `submission` can affect serialization as `application/xml`. Given the following XForms fragment:

```
<xforms:model xmlns:xforms="http://www.w3.org/2002/xforms"
              xmlns:my="http://ns.example.org/2003">
  <xforms:instance>
    <qname xmlns="">my:sample</qname>
  </xforms:instance>
  <xforms:submission method="post" action="..."/>
</xforms:model>
```

Note that the `includenamespaceprefixes` attribute is not present, which causes all namespace nodes to be serialized, resulting in the following serialized instance data:

```
<qname xmlns:xforms="http://www.w3.org/2002/xforms"
       xmlns:my="http://ns.example.org/2003">my:sample</qname>
```

In particular, note that the XForms namespace has been serialized. To prevent this example from including the unneeded XForms namespace while maintaining the needed `my` prefix, `includenamespaceprefixes="my"` must be added to the submission element. When this attribute is present, the author takes responsibility to list all namespace prefixes not visibly utilized by the submitted instance data.

The following attributes correspond (in spelling, processing, and default values) to attributes on the `output` element of [XSLT 1.0], with the exception of using `xsd:boolean` to replace `"yes"|"no"`:

version
indent
encoding
omit-xml-declaration
cdata-section-elements

**Note:**

The following XSLT attributes have no counterpart in XForms:

doctype-system
doctype-public

Elements defined in the XForms Actions module, when that module is included, are also allowed in the content model of `submission`.

### 3.3.4 The bind Element

Element `bind` selects a node-set selected from the instance data with a model binding expression in the `nodeset` attribute. Other attributes on element `bind` encode model item properties to be applied to each node in the node-set. When `bind` has an attribute of type `xsd:ID`, the `bind` then associates that identifier with the selected node-set.

Common Attributes: Common, Model Item Properties

Special Attributes:

nodeset

A model binding expression that selects the set of nodes on which this `bind` operates, as defined in **7.5.2 Model Binding Expressions**.

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 **9.3.5 The insert Element**.

See **7.4 Evaluation Context** for details on how binding affects the evaluation context.

## 3.4 The XForms MustUnderstand Module

Certain elements, such as `extension` or foreign namespaced elements defined in a host language might be critical to the operation of a particular form. To indicate this, the MustUnderstand module defines a single attribute that can be used on any element.

| Element | Attributes | Minimal Content Model |
|---|---|---|
| *ANY* | xforms:mustUnderstand (xsd:boolean) | *n/a* |

It is a terminating error that must be reported to the user if an element is marked `mustUnderstand="true"`, and the XForms Processor does not have an implementation available for processing the element.

## 3.5 The XForms Extension Module

There are many different ways a host language might include XForms. One approach uses only well-formed processing, disregarding validation. Another case uses strict validation, for example XHTML 1.0, in which only predefined elements are allowed. Another common approach is to allow unregulated content in a few selected places. A host language that chooses this option can use the Extension module.

| Element | Attributes | Minimal Content Model |
|---|---|---|
| extension | Common | ANY |

### 3.5.1 The extension Element

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.

Common Attributes: Common

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

```
<input ref="dataset/user/email" id="email-input">
  <label>Enter your email address</label>
  <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>
```

# 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.

State transitions are in general initiated by sending events to parts of the XForms tree. The XForms Processing Model consists of events in the following categories:

- Initialization
- Interaction
- Notification
- Error Conditions

## 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][XML Events], with an event capture phase, arrival of the event at its Target, and finally the event bubbling phase.

Throughout this chapter, each reference to "form control" as a target element is a shorthand for any of the following elements: `input`, `secret`, `textarea`, `output`, `upload`, `trigger`, `range`, `submit`, `select`, `select1`, or `group`.

| Event name | Cancelable? | Bubbles? | Target element |
|---|---|---|---|
| 4.2 Initialization Events | | | |
| xforms-model-construct | No | Yes | `model` |
| xforms-model-construct-done | No | Yes | `model` |
| xforms-ready | No | Yes | `model` |
| xforms-model-destruct | No | Yes | `model` |
| 4.3 Interaction Events | | | |

| Event name | Cancelable? | Bubbles? | Target element |
|---|---|---|---|
| xforms-previous | Yes | No | form control |
| xforms-next | Yes | No | form control |
| xforms-focus | Yes | No | form control |
| xforms-help | Yes | Yes | form control |
| xforms-hint | Yes | Yes | form control |
| xforms-rebuild | Yes | Yes | `model` |
| xforms-refresh | Yes | Yes | `model` |
| xforms-revalidate | Yes | Yes | `model` |
| xforms-recalculate | Yes | Yes | `model` |
| xforms-reset | Yes | Yes | `model` |
| xforms-submit | Yes | Yes | `submission` |
| **4.4 Notification Events** | | | |
| DOMActivate | Yes | Yes | form control |
| xforms-value-changed | No | Yes | form control |
| xforms-select | No | Yes | `item` or `case` |
| xforms-deselect | No | Yes | `item` or `case` |
| xforms-scroll-first | No | Yes | `repeat` |
| xforms-scroll-last | No | Yes | `repeat` |
| xforms-insert | No | Yes | `instance` |
| xforms-delete | No | Yes | `instance` |
| xforms-valid | No | Yes | form control |
| xforms-invalid | No | Yes | form control |
| DOMFocusIn | No | Yes | form control |
| DOMFocusOut | No | Yes | form control |
| xforms-readonly | No | Yes | form control |
| xforms-readwrite | No | Yes | form control |
| xforms-required | No | Yes | form control |
| xforms-optional | No | Yes | form control |
| xforms-enabled | No | Yes | form control |
| xforms-disabled | No | Yes | form control |
| xforms-in-range | No | Yes | form control |
| xforms-out-of-range | No | Yes | form control |
| xforms-submit-done | No | Yes | `submission` |
| xforms-submit-error | No | Yes | `model` |
| **4.5 Error Indications** | | | |
| xforms-binding-exception | No | Yes | any element that can contain a binding expression |
| xforms-link-exception | No | Yes | `model` |
| xforms-link-error | No | Yes | `model` |
| xforms-compute-exception | No | Yes | `model` |

## 4.2 Initialization Events

This section defines the various stages of the *initialization* phase. The processor begins initialization by dispatching an event `xforms-model-construct` to each XForms Model in the containing document.

### 4.2.1 The xforms-model-construct Event

Dispatched by the containing document processor to bootstrap XForms Processor initialization.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following:

1 All XML Schemas loaded. If an error occurs while attempting to access or process a remote document, processing halts with an exception (**4.5.2 The xforms-link-exception Event**).

2 If an external source for the initial instance data is given, an XPath data model [**7 XPath Expressions in XForms**] is constructed from it; otherwise if inline initial instance data is given, that is used instead. If the external initial instance data is not well-formed XML or cannot be retrieved, processing halts with an exception (**4.5.2 The xforms-link-exception Event**). If neither are given, the data model is not constructed in this phase, but during user interface construction (**4.2.2 The xforms-model-construct-done Event**).

3 If applicable, P3P initialized. [P3P 1.0]

4 Instance data is constructed. All strings inserted into the instance data are subject to Unicode normalization. All model item properties are initialized by processing all `bind` elements in document order. For each `bind`:

   a The attribute `nodeset` attached to the bind is evaluated, resulting in a set of nodes selected.

   b For each node in the node-set, model item properties are applied according to the remaining attributes on `bind`: the string value of each attribute (with a name matching one of the properties defined in **6.1 Model Item Property Definitions**) is copied as the local value of the model item property of the same name.

   c If the node already contains a model item property of the same name, XForms processing for this containing document halts with an exception (**4.5.1 The xforms-binding-exception Event**).

5 Perform an `xforms-rebuild`, `xforms-recalculate`, and `xforms-revalidate` in sequence, for this `model` element. (The `xforms-refresh` is not performed since the user interface has not yet been initialized).

After all XForms Models have been initialized, an `xforms-model-construct-done` event is dispatched to each `model` element.

### 4.2.2 The xforms-model-construct-done Event

Dispatched after the completion of `xforms-model-construct` processing.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event happens once, no matter how many XForms Models are present in the containing document, and results in the following, for each form control:

Processing can proceed in one of two different ways depending on whether an `instance` in a `model` exists when the first form control is processed.

If the `instance` referenced on the form control existed when the first form control was processed:

1 The binding expression is evaluated to ensure that it points to a node that exists. If this is not the case then the form control should behave in the same manner as if it had bound to a model item with the `relevant` model item property resolved to `false`.

If the `instance` referenced on the form control did not exist when the first form control for the same `instance` was processed:

1 For the first reference to an `instance` a default `instance` is created by following the rules described below.

   a A root `instanceData` element is created.

   b An instance data element node will be created using the binding expression from the user interface control as the `name`. If the `name` is not a valid QName, processing halts with an exception (**4.5.1 The xforms-binding-exception Event**).

2 For the second and subsequent references to an `instance` which was automatically created the following processing is performed:

   a If a matching instance data node is found, the user interface control will be connected to that element.

b If a matching instance data node is not found, an instance data node will be created using the binding expression from the user interface control as the `name`. If the `name` is not a valid QName, processing halts with an exception (**4.5.1 The xforms-binding-exception Event**).

After all form controls have been initialized, an `xforms-ready` event is dispatched to each `model` element.

### 4.2.3 The xforms-ready Event

Dispatched as part of `xforms-model-construct-done` processing.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

### 4.2.4 The xforms-model-destruct Event

Dispatched by the processor to advise of imminent shutdown of the XForms Processor, which can occur from user action, or from the `load` XForms Action, or as a result of form submission.

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

## 4.3 Interaction Events

### 4.3.1 The xforms-next and xforms-previous Events

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

Target: form control

Bubbles: No

Cancelable: Yes

Context Info: None

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

Navigation is determined on a containing document-wide basis. The host language is responsible for defining overall navigation order. The following describes a possible technique based on a `navindex` attribute, using individual form controls as a navigation unit: 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. Form controls that have a `navindex` specified and assign a positive value to it are navigated first.
   a. Outermost form controls are navigated in increasing order of the `navindex` value. Values need not be sequential nor must they begin with any particular value. Form controls that have identical `navindex` values are to be navigated in document order.
   b. Ancestor form controls (<`group`>, <`repeat`>, and <`switch`>) establish a local navigation sequence. All form controls within a local sequence are navigated, in increasing order of the `navindex` value, before any outside the local sequence are navigated. Form controls that have identical `navindex` values are navigated in document order.
2. Those form controls that do not specify `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.

### 4.3.2 The xforms-focus Event

Dispatched in response to: set focus to a form control.

Target: form control

Bubbles: No

Cancelable: Yes

Context Info: None

The default action for these events results in the following:

focus is given to the target form control if the form control is able to accept focus.

### 4.3.3 The xforms-help and xforms-hint Events

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

Target: form control

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for these events results in the following: If the form control has help/hint elements supplied, these are used to construct a message that is displayed to the user. Otherwise, user agents may provide default help or hint messages, but are not required to.

### 4.3.4 The xforms-refresh Event

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

The default action for this event results in the following: The user interface reflects the state of the model, which means that all forms controls reflect for their corresponding bound instance data:

- its current value
- its validity
- whether it is `required`, `readonly` or `relevant`.

### 4.3.5 The xforms-revalidate Event

Dispatched in response to: a request to revalidate a particular XForms Model.

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

The default handling for this event must satisfy the following conditions:

1 All instance data nodes in all `instance` elements in the `model` are checked against any specified XML Schema.

2 All instance data nodes in all `instance` elements in the `model` are checked against any bound model item properties which define constraints on the value, i.e. `required`, `constraint` (**6 Model Item Properties**).

3 The appropriate notification events (**4.4.6 The xforms-valid Event**, **4.4.7 The xforms-invalid Event**, **4.4.10 The xforms-readonly Event**, **4.4.11 The xforms-readwrite Event**, **4.4.12 The xforms-required Event**, **4.4.13 The xforms-optional Event**, **4.4.14 The xforms-enabled Event**, **4.4.15 The xforms-disabled Event**) are dispatched to form controls where the matching model item property evaluates to a different value than at the start of the processing of this event.

**Note:**

Prior to the dispatching of the `xforms-ready` event handler, there are no form controls bound to instance data, so `xforms-valid` and other notification events are not dispatched.

### 4.3.6 The xforms-recalculate Event

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

The default action for this event results in the following:

The values of all instance data items match their associated 'calculate' constraints, if any. All model item properties that can contain computed expressions are resolved.

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. References to the current node's value in `calculate` expressions 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.

> **Note:**
>
> Authors should not refer to the current node's value in `calculate` expressions because the effect is not well-defined. Other model item properties, such as `required` or `readonly`, however, are well-defined in the presence of self-references.

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 should not recalculate computes that are not computationally dependent on one or more of the elements in *L*.

2 An XForms Processor should perform only 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 (**4.5.4 The xforms-compute-exception Event**).

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

### 4.3.7 The xforms-rebuild Event

Dispatched in response to: a request to rebuild the internal data structures that track computational dependencies within a particular XForms Model.

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

The default action for this event is that the computational dependency data structures are rebuilt, then the change list *L* is set to contain references to all instance nodes that have an associated computational expression such that a *full* recalculate is performed the next time the `xforms-recalculate` event is dispatched to the model.

### 4.3.8 The xforms-reset Event

Dispatched in response to: a user request to reset the model.

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

The instance data is reset to the tree structure and values it had immediately after having processed the `xforms-ready` event. Then, the events `xforms-rebuild`, `xforms-recalculate`, `xforms-revalidate` and `xforms-refresh` are dispatched to the `model` element in sequence.

### 4.3.9 The xforms-submit Event

See chapter **11 Submit**.

## 4.4 Notification Events

### 4.4.1 The DOMActivate Event

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

The default action for this event results in the following: None; notification event only.

### 4.4.2 The xforms-value-changed Event

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

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

**Note:**

For incremental processing, this specification does not define how often XForms Processors fire these events. Implementations are expected to optimize processing (for instance not flashing the entire screen for each character entered, etc.).

**Note:**

The change to the instance data associated with this event happens before the event is dispatched.

### 4.4.3 The xforms-select and xforms-deselect Events

Dispatched in response to: an item in a `select`, `select1`, or `switch` becoming selected or deselected.

Target: `item` or `case`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

### 4.4.4 The xforms-scroll-first and xforms-scroll-last Events

Dispatched in response to: a setindex action attempting to set an index outside the range of a `repeat`.

Target: `repeat`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

### 4.4.5 The xforms-insert and xforms-delete Events

Dispatched in response to: A event handler invoking an XForms Action `insert` or `delete`, successfully adding or deleting a repeat item..

Target: `instance`

Bubbles: Yes

Cancelable: No

Context Info: Path expression used for insert/delete (xsd:string).

The default action for these events results in the following: None; notification event only.

### 4.4.6 The xforms-valid Event

Dispatched in response to: an instance data node becoming valid.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes valid indirectly, through the `constraint` model item property evaluating to `true`.

### 4.4.7 The xforms-invalid Event

Dispatched in response to: an instance data node becoming invalid.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes invalid indirectly, through the `constraint` model item property evaluating to `false`.

### 4.4.8 The DOMFocusIn Event

Dispatched in response to: a form control receiving focus.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

#### 4.4.9 The DOMFocusOut Event

Dispatched in response to: a form control losing focus.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

#### 4.4.10 The xforms-readonly Event

Dispatched in response to: an instance data node becoming readonly.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes becomes indirectly, through the `readonly` model item property evaluating to `true`.

#### 4.4.11 The xforms-readwrite Event

Dispatched in response to: an instance data node becoming read-write.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes read-write indirectly, through the `readonly` model item property evaluating to `false`.

### 4.4.12 The xforms-required Event

Dispatched in response to: an instance data node becoming required.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes required indirectly, through the `required` model item property evaluating to `true`.

### 4.4.13 The xforms-optional Event

Dispatched in response to: an instance data node becoming optional.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes optional indirectly, through the `required` model item property evaluating to `false`.

### 4.4.14 The xforms-enabled Event

Dispatched in response to: an instance data node becoming enabled.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes enabled indirectly, through the `relevant` model item property evaluating to `true`.

### 4.4.15 The xforms-disabled Event

Dispatched in response to: an instance data node becoming disabled.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of the bound instance data node changes, additionally whenever the bound instance data node becomes disabled indirectly, through the `relevant` model item property evaluating to `false`.

### 4.4.16 The xforms-in-range Event

Dispatched in response to: the value of an instance data node has changed such that the value can now be represented by the form control.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of an instance data node that was not possible to represent given the constraints specified on a form control has changed such that the value can now be represented by the form control.

### 4.4.17 The xforms-out-of-range Event

Dispatched in response to: the value of an instance data node has changed such that the value can not be represented by the form control.

Target: form control

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

This event is dispatched whenever the value of an instance data node can not be represented given the constraints specified on a form control.

#### 4.4.18 The xforms-submit-done Event

Dispatched in response to: completion of submit processing, including processing any returned document.

Target: `submission`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: None; notification event only.

#### 4.4.19 The xforms-submit-error Event

Dispatched as an indication of: a failure of the submit process, as defined at **11 Submit**

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: The submit method URI that failed (xsd:anyURI)

The default action for this event results in the following: None; notification event only.

## 4.5 Error Indications

Error indications happen as a result of unusual conditions in the XForms Processor. Some of these are "fatal" errors, which halt processing, and bear the suffix "exception". Others are simply for notification, and bear the suffix "error". For all events in this section, it is permissible for the XForms Processor to perform some kind of default handling, for example logging error messages to a file.

#### 4.5.1 The xforms-binding-exception Event

Dispatched as an indication of: an illegal binding expression, or a `model` attribute that fails to point to the ID of a `model` element, or a `bind` attribute that fails to point to the ID of a `bind` element, or a `submission` attribute that fails to point to the ID of a `submission` element.

Target: any element that can contain a binding expression

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following: Fatal error.

### 4.5.2 The xforms-link-exception Event

Dispatched as an indication of: a failure in link traversal of a linking attribute.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: The URI that failed to load (xsd:anyURI)

The default action for this event results in the following: Fatal error.

### 4.5.3 The xforms-link-error Event

Dispatched as an indication of: a failure in link traversal of a linking attribute, in a situation not critical to form processing.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: The URI that failed to load (xsd:anyURI)

The default action for this event results in the following: None; notification event only.

### 4.5.4 The xforms-compute-exception Event

Dispatched as an indication of: an error occurring during XPath evaluation.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: Implementation-specific error string.

The default action for this event results in the following: Fatal error.

## 4.6 Event Sequencing

The previous sections describe processing associated with individual events. This section gives the overall sequence of related events that must occur in several common situations. In the following lists, events that may be fired more than once are prefixed with [n].

### 4.6.1 For `input`, `secret`, `textarea`, `range`, or `upload` Controls

- When the form control is interactively changed, and has the "incremental="true" setting, the event sequence described at **4.6.7 Sequence: Value Change with Focus Change** may be initiated at implementation dependent intervals.
- When the form control is interactively changed and does not have the "incremental=true" setting, no events are required to be dispatched, and thus no order is defined.
- When focus changes from the form control and the value has changed, the event sequence is as described at **4.6.7 Sequence: Value Change with Focus Change**.

### 4.6.2 For `output` Controls

- No event sequences are defined.

### 4.6.3 For `select` or `select1` Controls

- When a selection is interactively changed, and the form control has the "incremental="true" setting, the event sequence is described at **4.6.6 Sequence: Selection Without Value Change**, which may be followed immediately by the sequence described at **4.6.7 Sequence: Value Change with Focus Change**.
- When a selection is interactively changed, and the form control does not have the "incremental="true" setting, the event sequence is described at **4.6.6 Sequence: Selection Without Value Change**.
- When focus changes from the form control and the value has changed, the event sequence is as described at **4.6.7 Sequence: Value Change with Focus Change**.

### 4.6.4 For `trigger` Controls

- Activating the form control causes the event sequence defined at **4.6.8 Sequence: Activating a Trigger**.

### 4.6.5 For `submit` Controls

- Activating the form control causes the event sequence defined at **4.6.8 Sequence: Activating a Trigger**, followed immediately by the event sequence defined at **4.6.9 Sequence: Submission**.

### 4.6.6 Sequence: Selection Without Value Change

1. xforms-deselect
2. xforms-select

### 4.6.7 Sequence: Value Change with Focus Change

1 xforms-recalculate

2 xforms-revalidate

3 [n] xforms-valid/xforms-invalid; xforms-enabled/xforms-disabled; xforms-optional/xforms-required; xforms-readonly/xforms-readwrite

4 xforms-value-changed

5 DOMFocusOut

6 DOMFocusIn

7 xforms-refresh

Reevaluation of binding expressions must occur before step 3 above.

### 4.6.8 Sequence: Activating a Trigger

1 DOMActivate

### 4.6.9 Sequence: Submission

1 xforms-submit

2 xforms-submit-done or xforms-submit-error

# 5 Datatypes

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

## 5.1 XML Schema Built-in Datatypes

XForms supports all XML Schema datatypes except for `xsd:duration`, `xsd:ENTITY`, `xsd:ENTITIES`, and `xsd:NOTATION`. Concepts value space, lexical space and constraining facets are as specified in [XML Schema part 2]. Certain XML Schema datatypes have been identified as part of a smaller XForms conformance profile that is being developed separately, and are marked with an asterisk *. XForms includes datatypes *derived by restriction* and *derived by list* from these base types. XForms Processors must treat the datatypes listed in the chapter as in-scope without requiring the inclusion of an XML Schema.

Built-in primitive types:

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

**Note:**

The built-in datatype `xsd:duration` is not supported, except as an abstract datatype. Instead, either `xforms:dayTimeDuration` or `xforms:yearMonthDuration` should be used.

Built-in derived types:

normalizedString
token
language
Name
NCName
ID
IDREF
IDREFS
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.

### 5.2.1 xforms:listItem

This datatype serves as a base for the `xforms:listItems` datatype. The value space for listItem permits one or more characters valid for xsd:string, except white space 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 **9.3.3 The itemset Element**.

### 5.2.3 xforms:dayTimeDuration

XForms includes a totally ordered duration datatype that can represent a duration of days, hours, minutes, and fractional seconds. The value space for this datatype is the set of fractional second values. This datatype is derived from `xsd:duration`.

### 5.2.4 xforms:yearMonthDuration

XForms includes a totally ordered duration datatype that can represent a duration of a whole number of months and years. The value space for this datatype is the set of integer month values. This datatype is derived from `xsd:duration`.

# 6 Model Item Properties

This chapter defines infoset contributions that can be bound to instance data nodes with element `bind` (see **3.3.4 The bind Element**). The combination of these contributions to an instance data node is called a model item. Taken together, these contributions are called model item properties, and are defined in the following section. In contrast, the term Schema constraint refers only to XML Schema constraints from the facets of a given datatype.

## 6.1 Model Item Property Definitions

Model item properties can be distinguished along various axes.

Computed expressions vs. fixed properties:

- Fixed properties are static values that the XForms Processor evaluates only once. Such properties consist of literals, and are not subject to XPath evaluation.
- Computed expressions are XPath expressions that provide a value to the XForms Processor. Such values are recalculated at certain times as specified by the XForms Processing Model (see **4 Processing Model**). These expressions encode dynamic properties, often 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.

Inheritance rules:

Some model item properties define inheritance rules, in which case the XForms Processor needs to keep track of two separate values: 1) the **local value**, which is applied from an attribute of element `bind`, and 2) the **inherited value**, which is determined by combining the evaluated local value with the evaluated values from ancestor nodes in the instance data.

**Note:**

The sample recalculation algorithm defined in **D Recalculation Sequence Algorithm** is defined to operate only on the local values of a model item property. It assumes that an implementation propagates the combined values to a node's descendants.

Assigning local values:

Local values are assigned by processing all bind elements in an XForms Model in document order. It is an error to attempt to set a model item property twice on the same node. The details of this process are given at **4.2.1 The xforms-model-construct Event**.

The following sections list the model item properties available as part of all model items. For each, the following information is provided:

Description
Computed Expression (yes or no)
Legal Values
Default Value
Inheritance Rules

### 6.1.1 The type Property

Description: associates a Schema datatype.

Computed Expression: No.

Legal Values: Any `xsd:QName` representing a datatype definition in an XML Schema.

Default Value: `xsd:string`.

Inheritance Rules: does not inherit.

The effect of this model item property is the same as placing attribute `xsi:type` on the instance data. However, in contrast to `xsi:type`, `type` can be added to both elements and attributes.

Example: Attaching a XML Schema type constraint

```
<instance>
  <my:person-name>
    <my:first-name />
    <my:last-name xsi:type="my:nonEmptyString" />
  </my:person-name>
</instance>
<bind type="my:nonEmptyString" nodeset="/my:person-name/my:first-name" />
```

Here, we have illustrated two ways in which an XML Schema type can be associated with an element.

### 6.1.2 The readonly Property

Description: describes whether the value is restricted from changing.

Computed Expression: Yes.

Legal Values: Any expression that is convertible to XPath `boolean` with `boolean()`.

Default Value: `false()`, unless a `calculate` property is specified, then `true()`.

Inheritance Rules: If any ancestor node evaluates to `true`, this value is treated as `true`. Otherwise, the local value is used.

> **Note:**
>
> This is the equivalent of taking the logical OR of the evaluated `readonly` property on the local and every ancestor node.

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

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

Example: Attaching a readonly property

```
<instance>
  <my:person-name>
    <my:first-name>Roland</my:first-name>
    <my:last-name/>
  </my:person-name>
</instance>
<bind nodeset="/my:person-name/my:first-name" readonly="true()"/>
```

Here, we have associated a `readonly` property with an element.

### 6.1.3 The required Property

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

Computed Expression: Yes.

Legal Values: Any expression that is convertible to XPath `boolean` with `boolean()`.

Default Value: `false()`.

Inheritance Rules: does not inherit.

A form may *require* certain values, and this requirement may be dynamic. When evaluating to `true`, this model item property 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` model item property 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.

Example: Attaching a required property

```
<instance>
  <my:person-name>
    <my:first-name>Roland</my:first-name>
    <my:last-name />
  </my:person-name>
</instance>
<bind nodeset="/my:person-name/my:last-name" required="true()"/>
```

Here, we have associated a `required` property with element `my:last-name` to indicate that a value must be supplied.

**Note:**

XML Schema has a similarly named concept with `use="required|optional|prohibited"`. This is different than the XForms Model item property, in two ways: 1) `use` applies only to attributes, while XForms `required` applies to any node. 2) `use` is concerned with whether the entire attribute must be specified (without regard to value), while `required` determines whether a value is required of the node before submission.

### 6.1.4 The relevant Property

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

Computed Expression: Yes.

Legal Values: Any expression that is convertible to XPath `boolean` with `boolean()`.

Default Value: `true()`.

Inheritance Rules: If any ancestor node evaluates to XPath `false`, this value is treated as `false`. Otherwise, the local value is used.

**Note:**

This is the equivalent of taking the logical AND of the evaluated `relevant` property on the local and every ancestor node.

Many forms have data entry sections 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.

The `relevant` model item property 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.

Example: Attaching a relevant property

```
<instance>
  <my:order>
    <my:item>
      <my:amount />
      <my:discount>100</my:discount>
    </my:item>
  </my:order>
</instance>
<bind nodeset="my:item/my:discount" readonly="true()"
      relevant="../my:amount &gt; 1000"/>
```

Here, we have associated a `relevant` property with element `my:discount` to indicate a discount is relevant when the order amount is greater than 1000.

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

| | `required="true()"` | `required="false()"` |
|---|---|---|
| `relevant="true()"` | The form control (and any children) must be visible or available to the user. The XForms user interface may indicate that a value is required. | The form control (and any children) must be visible or available to the user. The XForms user interface may indicate that a value is optional. |
| `relevant="false()"` | The form control (and any children) must be hidden or unavail- | The form control (and any children) must be hidden or |

| | | |
|---|---|---|
| | able to the user. Entering a value or obtaining focus should not be allowed. The XForms user interface may indicate that should the form control become relevant, a value would be required. | unavailable to the user. Entering a value or obtaining focus should not be allowed. |

### 6.1.5 The calculate Property

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

Computed Expression: Yes.

Legal Values: Any XPath expression.

Default Value: none.

Inheritance Rules: does not inherit.

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. Chapter **4 Processing Model** contains details of when and how the calculation is performed.

Example: Attaching a calculate property

```
<instance>
  <my:order>
    <my:item>
      <my:amount />
      <my:discount />
    </my:item>
  </my:order>
</instance>
<bind nodeset="my:item/my:discount" calculate="../my:amount * 0.1"
      relevant="../my:amount &gt; 1000"/>
```

Here, we have associated a `relevant` property with element `my:discount` to indicate a discount of 10% is relevant when the order amount is greater than 1000.

### 6.1.6 The constraint Property

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

Computed Expression: Yes.

Legal Values: Any expression that is convertible to XPath `boolean` with `boolean()`.

Default Value: `true()`.

Inheritance Rules: does not inherit.

When evaluating to XPath `false`, the associated model item is not valid; the converse is not necessarily true. Chapter **4 Processing Model** contains details of when and how the constraint is calculated as well as when validation is performed.

Example: Attaching a constraint property

```
<instance>
  <my:range>
    <my:from />
    <my:to />
  </my:range>
</instance>
<bind nodeset="my:to" constraint=". &gt; ../my:from" />
```

Here, a `constraint` property associated with element `my:to` indicates that its value must be greater than that of element `my:from`.

**Note:**

Specifying minimum and maximum occurrences for nodes in the instance data can be achieved by using the `count()` function within a `constraint` property.

### 6.1.7 The p3ptype Property

Description: Attaches a P3P data element to an instance data node, indicating the specific kind of data collected there.

Computed Expression: No.

Legal Values: `xsd:string`.

Default Value: none

Inheritance Rules: does not inherit.

This model item property holds a description of the kind of data collected by the associated instance data node, based on the P3P datatype system [P3P 1.0]. This information may be used to enhance the form-fill experience, for example by supplying previously-known data.

Example: Attaching a type constraint using Binding

```
<instance>
  <my:person-name>
    <my:first-name />
    <my:last-name />
  </my:person-name>
</instance>
<bind type="my:nonEmptyString" nodeset="my:first-name"
      p3ptype="user.personname.given"/>
```

Here, we have attached both XML Schema and P3P type information to element `first-name` via element `bind`.

## 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 XForms Model. 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.

### 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 an XML 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 associated with a model item must be determined by the following list, as if it were processed in the given order. When multiple datatype restrictions apply to the same model item, the combination of all given restrictions must apply. Note that it is possible to produce a combination of restrictions that is impossible to satisfy; authors are encouraged to avoid this practice.

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 instance data 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 XML 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="my:nonEmptyString"/>
```

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

Example: Attaching Type Constraint Using XForms Binding

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

Here, we have attached type information to element `first-name` via element `bind`. Thus the XForms author can extend external schemas without having the ability to change them.

# 7 XPath Expressions in XForms

XForms uses XPath to address instance data nodes in binding expressions, to express constraints, and to specify calculations. XPath expressions that are not syntactically valid, including attempted calls to undefined functions, result in an exception (**4.5.4 The xforms-compute-exception Event**), except for binding expressions, which produce a different exception (**4.5.1 The xforms-binding-exception Event**).

## 7.1 XPath Datatypes

XPath datatypes 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 Feature string for the hasFeature method call

For this version of the XForms specification, the feature string for the [DOM2 Core] `DOMImplementation` interface `hasFeature` method call is `"org.w3c.xforms.dom"` and the version string is `"1.0"`.

## 7.3 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]. XForms Processors that implement DOM must provide DOM access to this instance data via the interface defined below.

**Note:**

Instance data always has a single root element, and thus corresponds to a DOM Document.

The IDL for this interface follows:

```
#include "dom.idl"
pragma prefix "w3c.org"
module xforms {
  interface XFormsModelElement : dom::Element {
    dom::Document getInstanceDocument(in dom::DOMString instanceID)
      raises(dom::DOMException);
    void rebuild();
    void recalculate();
    void revalidate();
    void refresh();
  };
};
```

### 7.3.1 The getInstanceDocument() Method

This method returns a DOM Document that corresponds to the instance data associated with the `instance` element containing an `ID` matching the `instance-id` parameter. If there is no matching instance data, a `DOMException` is thrown.

### 7.3.2 The rebuild() Method

This method signals the XForms Processor to rebuild any internal data structures used to track computational dependencies within this XForms Model. This method takes no parameters and raises no exceptions.

### 7.3.3 The recalculate() Method

This method signals the XForms Processor to perform a full recalculation of this XForms Model. This method takes no parameters and raises no exceptions.

**Note:**

Script invocation of `recalculate()` is not necessarily equivalent to performing the recalculate action handler. Though the script is assumed to have modified instance data prior to invoking `recalculate()`, the DOM mutations are not cached. Thus, a *full* recalculation is necessary to ensure the proper changes are effected throughout the XForms Model.

### 7.3.4 The revalidate() Method

This method signals the XForms Processor to perform a full revalidation of this XForms Model. This method takes no parameters and raises no exceptions.

### 7.3.5 The refresh() Method

This method signals the XForms Processor to perform a full refresh of form controls bound to instance nodes within this XForms Model. This method takes no parameters and raises no exceptions.

## 7.4 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 XPath expressions as part of XForms:

1 The context node for outermost binding elements is the top-level element node, or the single node returned by `/*`. A **binding element** is any element 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 always resides within the context model, which is determined choosing the first item that applies from this list:

   a If a `model` attribute is present on the binding element, the attribute determines the context model.

   b If the binding element has an immediately enclosing binding element, the context model of the immediately enclosing binding element is used.

   c The first model in document order is used.

4 The context node for computed expressions (occurring on element bind) is the node currently being processed.

5 For Single-Node binding expressions, the context size and position are 1. For Nodeset binding expressions, the context size is the size of the node-set, and the context position is the document order position of the node currently being processed within the node-set.

6 No variable bindings are in place.

7 The available function library is defined below, plus any functions supplied by the implementation. Extension functions required for operation of the form should be declared, as described at **7.12 Extension Functions**.

8 Any namespace declarations in scope for the attribute that defines the expression are applied to the expression.

Example: Binding Expression Context Nodes

```
<group ref="level2/level3">
  <select1 ref="@attr" ... />
</group>
```

In this example, the `group` has a binding expression of `level2/level3`. According to the rules above, this outermost element node would have a context node of `/level1`, which is the top-level element node of the instance data. The `select1` form control then inherits a context node from the parent group. Matching instance data, represented as serialized XML, follows:

Example: Sample XML Instance Data

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

## 7.5 Binding Expressions

A binding expression is an XPath PathExpr used in binding a model item property to one or more instance nodes, or to bind a form control to instance data, or to specify the node or node set for operation by an action. By default, all binding expressions refer to the first instance within the context model. This behavior can be changed with the `instance()` function.

### 7.5.1 Dynamic Dependencies

Not every possible XPath expression is acceptable as a binding expression. In particular, there are restrictions on model binding expressions that create **dynamic dependencies**, which are defined as follows:

An XPath predicate (in square brackets) is a possibly implicit boolean test. A dynamic dependency exists on any predicate unless all terms in the test are "fixed", where fixed means either a constant, or a value that will not change between operations explicitly defined as rebuilding computational dependencies.

> **Note:**
>
> For purposes of determining dynamic dependencies, the following subexpressions are considered fixed: `position()`, `last()`, `count()`, and `property()`. This is because the specification mandates a dependency rebuild after any event that could change the values returned by these functions.

Another dynamic dependency is any use of the `id()` function, unless both the parameter to the function and the matching attribute of type `xsd:ID` are fixed. In the same way, the `instance()` function is dynamic unless the parameter to the function is fixed.

XPath variables that change in value from one recalculate to the next would also create dynamic dependencies (though XForms 1.0 defines an empty variable context for all XPath expressions).

Authors that define extension functions are encouraged to follow these rules.

### 7.5.2 Model Binding Expressions

A model binding expression is a kind of binding expression that can be used to declare model item properties, and is used in attributes of the `bind` element.

Dynamic dependencies in model binding expressions will generally require manual rebuilding of dependencies.

### 7.5.3 UI Binding Expressions

Binding references can be used to bind form controls to the underlying instance data as described here. Different attribute names, `ref` and `nodeset` distinguish between a single node and a node-set respectively. See **3.2.3 Single-Node Binding Attributes** and **3.2.4 Node-Set Binding Attributes**.

Dynamic dependences are allowed in UI binding expressions based on the conformance profile.

### 7.5.4 UI Binding in other XML vocabularies

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 XHTML 1.x user interface controls as shown below. See **3.2.3 Single-Node Binding Attributes** and **3.2.4 Node-Set Binding Attributes**.

Example: XForms Binding In XHTML 1.x User Interface Controls

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

### 7.5.5 Binding Examples

Consider the following document with the one-and-only XForms model:

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

## 7.6 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.

These following sections define additional required functions for use within XForms.

## 7.7 Boolean Functions

### 7.7.1 The boolean-from-string() Function

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

Function `boolean-from-string` returns `true` if the required parameter `string` is "true" or "1", or `false` if parameter `string` is "false", or "0". This is useful when referencing a Schema `xsd:boolean` datatype in an XPath expression. If the parameter string matches none of the above strings, according to a case-insensitive comparison, processing stops with an exception (**4.5.4 The xforms-compute-exception Event**).

### 7.7.2 The if() Function

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

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

## 7.8 Number Functions

### 7.8.1 The avg() Function

*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()`. If the parameter is an empty node-set, the return value is `NaN`.

### 7.8.2 The min() Function

*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, or if any of the nodes evaluate to `NaN`, the return value is `NaN`.

### 7.8.3 The max() Function

*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, or if any of the nodes evaluate to `NaN`, the return value is `NaN`.

### 7.8.4 The count-non-empty() Function

*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.

### 7.8.5 The index() Function

*number* **index**(*string*)

Function `index` takes a string argument that is the `IDREF` of a `repeat` and returns the current 1-based position of the repeat index for the identified `repeat`—see **9.3.1 The repeat Element** for details on `repeat` and its associated repeat index. If the specified argument does not identify a `repeat`, processing stops with an exception (**4.5.4 The xforms-compute-exception Event**).

Example: index

```
<xforms:trigger>
  <xforms:label>Add to Shopping Cart</xforms:label>
  <xforms:insert ev:event="DOMActivate" position="after"
                 nodeset="items/item" at="index('cartUI')"/>
</xforms:trigger>
```

## 7.9 String Functions

### 7.9.1 The `property()` Function

*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 **12 Conformance**.

Example: property

```
<xforms:instance>
  ...
  <xforms:bind nodeset="message"
               calculate="concat( 'created with XForms ', property('version'))"/> ...
</xforms:instance>
```

## 7.10 Date and Time Functions

**Note:**

The following XML Schema datatypes do not have specific functions for manipulation within XForms expressions: `xsd:time`, `xsd:gYearMonth`, `xsd:gYear`, `xsd:gMonthDay`, `xsd:gDay`, `xsd:gMonth`. Extension functions (**7.12 Extension Functions**) may be used to perform needed operations on these datatypes.

### 7.10.1 The now() Function

*string* **now**()

The `now` function returns the current system date and time as a string value in the canonical XML Schema `xsd:dateTime` format. If time zone information is available, it is included (normalized to UTC). If no time zone information is available, an implementation default is used.

**Note:**

Attaching a calculation of "`now()`" to an instance data node would not result in a stream of continuous recalculations of the XForms Model.

### 7.10.2 The days-from-date() Function

*number* **days-from-date**(*string*)

This function returns a whole number of days, according to the following rules:

If the string parameter represents a legal lexical `xsd:date` or `xsd:dateTime`, the return value is equal to the number of days difference between the specified date and `1970-01-01`. Hour, minute, and second components are ignored. Any other input parameter causes a return value of `NaN`.

Examples:

`days-from-date("2002-01-01")` returns `11688`
`days-from-date("1969-12-31")` returns `-1`

### 7.10.3 The seconds-from-dateTime() Function

*number* **seconds-from-dateTime**(*string*)

This function returns a possibly fractional number of seconds, according to the following rules:

If the string parameter represents a legal lexical `xsd:dateTime`, the return value is equal to the number of seconds difference between the specified dateTime and `1970-01-01T00:00:00Z`. If no time zone is specified, an implementation default is used. Any other input string parameter causes a return value of `NaN`.

### 7.10.4 The seconds() Function

*number* **seconds**(*string*)

This function returns a possibly fractional number of seconds, according to the following rules:

If the string parameter represents a legal lexical `xsd:duration`, the return value is equal to the number specified in the seconds component plus 60 * the number specified in the minutes component, plus 60 * 60 * the number specified in the hours component, plus 60 * 60 * 24 * the number specified in the days component. The sign of the result will match the sign of the duration. If no time zone is specified, an implementation default is used. Year and month components, if present, are ignored. Any other input parameter causes a return value of `NaN`.

Examples:

> `seconds("P1Y2M")` returns `0`
> `seconds("P3DT10H30M1.5S")` returns `297001.5`
> `seconds("3")` returns `NaN`

> **Note:**
>
> Even though this function is defined based on a lexical `xsd:duration`, it is intended for use only with derived-from-`xsd:duration` datatypes, specifically `xforms:dayTimeDuration`.

### 7.10.5 The months() Function

*number* **months**(*string*)

This function returns a whole number of months, according to the following rules:

If the string parameter represents a legal lexical `xsd:duration`, the return value is equal to the number specified in the months component plus 12 * the number specified in the years component. The sign of the result will match the sign of the duration. Day, hour, minute, and second components, if present, are ignored. Any other input parameter causes a return value of `NaN`.

Examples:

`months("P1Y2M")` returns `14`
`months("-P19M")` returns `-19`

**Note:**

Even though this function is defined based on a lexical `xsd:duration`, it is intended for use only with derived-from-`xsd:duration` datatypes, specifically `xforms:yearMonthDuration`.

## 7.11 Node-set Functions

### 7.11.1 The instance() Function

*node-set* **instance**(*string*)

An XForms Model can contain more that one instance. This function allows access to instance data, within the same XForms Model, but outside the instance data containing the context node.

The argument is converted to a string as if by a call to the `string` function. This string is treated as an IDREF, which is matched against `instance` elements in the containing document. If a match is located, and the matching instance data is associated with the same XForms Model as the current context node, this function returns a node-set containing just the root element node (also called the document element node) of the referenced instance data. In all other cases, an empty node-set is returned.

Example:

For instance data corresponding to this XML:

```
<xforms:instance xmlns="" id="orderform">
  <orderForm>
    <shipTo>
      <firstName>John</firstName>
    </shipTo>
  </orderForm>
</xforms:instance>
```

The following expression selects the `firstName` node. Note that the `instance` function returns an element node, effectively replacing the leftmost location step from the path:

```
ref="instance('orderform')/shipTo/firstName"
```

## 7.12 Extension Functions

XForms documents may use additional XPath extension functions beyond those described here. A number of useful community extensions are defined at [EXSLT]. The names of any such extension functions must be declared in attribute `functions` 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 signaling an exception (**4.5.4 The xforms-compute-exception Event**) 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 to 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

## 8.1 The XForms Form Controls Module

Form controls are declared using markup elements, and their behavior refined via markup attributes.

| Element | Attributes | Minimal Content Model |
|---|---|---|
| input | Common, UI Common, Single Node Binding, inputmode (xsd:string), incremental (xsd:boolean) | label, (UI Common)* |
| secret | Common, UI Common, Single Node Binding, inputmode (xsd:string), incremental (xsd:boolean) | label, (UI Common)* |
| textarea | Common, UI Common, Single Node Binding, inputmode (xsd:string), incremental (xsd:boolean) | label, (UI Common)* |
| output | Common, Single Node Binding (optional), appearance ("full"\|"compact"\|"minimal"\|xforms:QNameButNotNCNAME), value (XPathExpression) | label? |
| upload | Common, UI Common, Single Node Binding, mediatype (xsd:string), incremental (xsd:boolean) | label, filename?, mediatype?, (UI Common)* |
| range | Common, UI Common, Single Node Binding, start (xsd:string), end (xsd:string), step (xsd:string), incremental (xsd:boolean) | label, (UI Common)* |
| trigger | Common, UI Common, Single Node Binding (optional) | label, (UI Common)* |
| submit | Common, UI Common, Single Node Binding (optional), submission (xsd:IDREF) | label, (UI Common)* |

| | | |
|---|---|---|
| select | Common, UI Common, Single Node Binding, selection ("open" \| "closed"), incremental (xsd:boolean) | label, (List UI Common)+, (UI Common)* |
| select1 | Common, UI Common, Single Node Binding, selection ("open" \| "closed"), incremental (xsd:boolean) | label, (List UI Common)+, (UI Common)* |
| choices | Common | label?, (List UI Common)+ |
| item | Common | label, value, (UI Common)* |
| filename | Common, Single Node Binding | EMPTY |
| mediatype | Common, Single Node Binding | EMPTY |
| value | Common, Single Node Binding (optional) | (PCDATA\|ANY)* |
| label | Common, Single Node Binding (optional), Linking | (PCDATA\|(UI Inline))* |
| help | Common, Single Node Binding (optional), Linking | (PCDATA\|(UI Inline))* |
| hint | Common, Single Node Binding (optional), Linking | (PCDATA\|(UI Inline))* |
| alert | Common, Single Node Binding (optional), Linking | (PCDATA\|(UI Inline))* |

See also: **9.3.3 The itemset Element**.

**Note:**

Unless bound to form controls, instance data nodes are not presented to the user; consequently, there is no need for a form control corresponding to HTML `input type="hidden"`.

The following attributes are common to many user-interface related XForms elements, here called the `UI Common` attribute group.

| Element | Attributes |
|---|---|
| (various) | appearance ("full"\|"compact"\|"minimal"\|QName-but-not-NCName) |

appearance

Optional attribute to define an appearance hint.

**Note:**

A host language is expected to add attributes such as `xml:lang` as well as an attribute, named `class`, that holds a list of strings that can be matched by CSS class selectors.

Further, a host language must provide a way to indicate overall navigation order among form controls and other elements included in the host language, as well as keyboard or direct access navigation to specific elements. One such proposal is to uses a pair of attributes named `navindex` and `accesskey`, defined as follows:

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 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.

The user agent must provide a means of identifying the accesskeys that can be used in a presentation. This may be accomplished in different ways by different implementations, for example through direct interaction with the application or via the user's guide. The accesskey requested by the author might not be made available by the player (for example it may not exist on the device used, or it may be used by the player itself). Therefore the user agent should make the specified key available, but may map the accesskey to a different interaction behavior.

Additionally, this module defines the following content sets:

| Content Set | Minimal Content Model |
|---|---|
| **UI Common** | (help\|hint\|alert\|Action)* |
| **List UI Common** | (choices\|item\|itemset)+ |
| **Form Controls** | (input\|secret\|textarea\|output\|upload\|range\|trigger\|submit\|select\|select1)* |
| **UI Inline** | (output)* |

As shown above, the XML Events module adds the Actions content set into the UI Common content set. A host language should add inline markup to the Inline content set. When the XForms Extension module is present, it too should be included in the UI Common content set.

### 8.1.1 Implementation Requirements Common to All Form Controls

XForms user interface controls are bound to the underlying instance data using binding attributes as defined in the chapter **6 Model Item Properties**.

Form controls enable accessibility by taking a uniform approach to such features as labels, help text, navigation, and keyboard shortcuts. Internationalization issues are addressed by following the same design principles as in XHTML. All form controls are suitable for styling as aural or visual media.

Form controls encapsulate high-level semantics without sacrificing the ability to deliver real implementations. For instance, the form control `select` enables the user to *select items from a set*. These form controls distinguish the functional aspects of the underlying control from the presentational and behavioral aspects. 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 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.

All form controls must meet the following implementation requirements:

- Form controls that write simpleContent to instance data must do so exactly as defined by the XForms Action **10.1.9 The setvalue Element**
- All form controls that read simpleContent instance data must do so as follows:

  Element nodes: if text child nodes are present, returns the string-value of the first text child node. Otherwise, returns "" (the empty string)

  Attribute nodes: returns the string-value of the node.

  Text nodes: returns the string-value of the node.

  Namespace, processing instruction, comment, and the XPath root node: behavior is undefined.

- Form controls must distinguish rendering between valid and invalid states. Control of this behavior should be made available to stylesheets.
- Form controls must indicate when the bound instance data contains a value the form control is not capable of rendering. Control of this behavior should be made available to stylesheets.
- Form controls must render upon request an explanation of the current state of a form control, including validity and associated model item properties. Control of this behavior should be made available to stylesheets.
- Form controls must provide a default explanation for the above when no user-specified explanation is available.

Sections in this chapter define the various form controls by specifying the following:

Description
Common Attributes
Special Attributes
Examples
Data Binding Restrictions
Implementation Requirements

### 8.1.2 The input Element

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

Common Attributes: Common, UI Common, Single Node UI Binding

Special Attributes:

inputmode

This form control accepts an input mode hint. **E Input Modes**.

incremental

when `true`, this form control will generate additional `xforms-value-changed` events. The default value for this attribute is `false`.

Example:

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

In the above, the `class` attribute can be used by a style sheet 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.

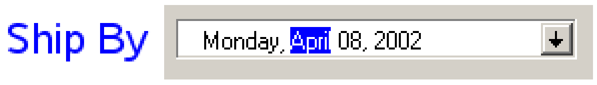
A graphical browser might render the above example as follows:

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

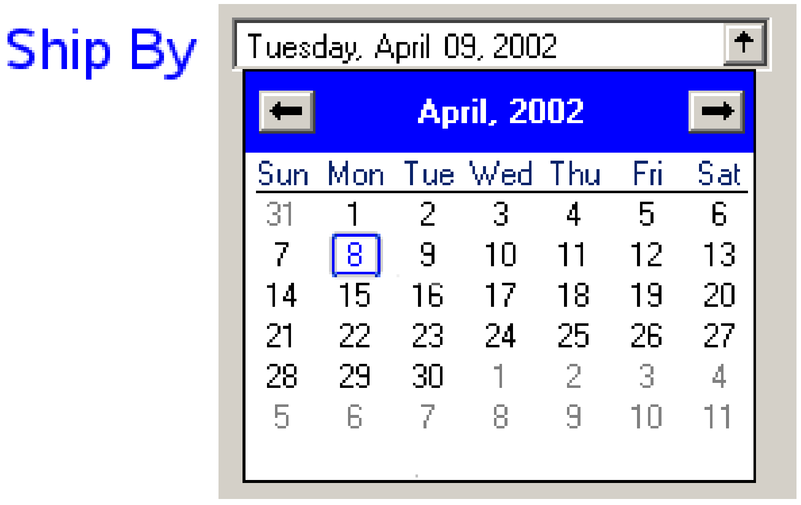
Implementation Requirements: Must allow entry of a lexical value for the bound datatype. Implementations should provide a convenient means for entry of datatypes and take into account localization and internationalization issues such as representation of numbers. For example, an `input` bound to an instance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

```
<input ref="order/shipDate">
  <label>Ship By</label>
  <hint>Please specify the ship date for this order.</hint>
</input>
```

A graphical browser might render the above example as follows:



The user can type a date into the text edit box, or press the button to open a calendar:



### 8.1.3 The secret Element

Description: This form control is used to provide the user with the ability to supply information to the system in a manner that makes it difficult for someone, other than the user, who may be observing the process to discern the value that is being supplied. A common use is for password entry.

Common Attributes: Common, UI Common, Single Node Binding

Special Attributes:

inputmode

This form control accepts an input mode hint. **E Input Modes**.

incremental

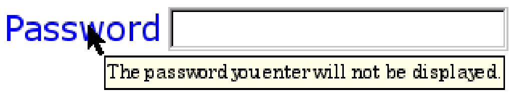
when `true`, this form control will generate additional `xforms-value-changed` events. The default value for this attribute is `false`.

Example:

Example: Password Entry

```
<secret ref="/login/password">
  <label>Password</label>
  <hint>The password you enter will not be displayed.</hint>
</secret>
```

A graphical browser might render this form control as follows:



Data Binding Restrictions: Identical to `input`.

Implementation Requirements: Implementations, including accessibility aids, must obscure the value being entered into this form control. One possible approach would be to render a "*" or similar character instead of the actual characters entered. Note that this provides only a casual level of security; truly sensitive information will require additional security measures outside the scope of XForms.

### 8.1.4 The textarea Element

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.

Common Attributes: Common, UI Common, Single Node Binding

Special Attributes:

inputmode

This form control accepts an input mode hint. **E Input Modes**.

incremental

when `true`, this form control will generate additional `xforms-value-changed` events. The default value for this attribute is `false`.

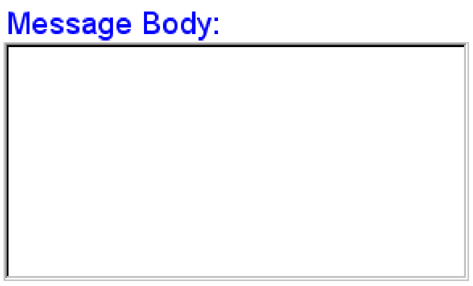
Example:

Example: Email Message Body

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

In the above, the `class` attribute can be used by a style sheet 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.

### 8.1.5 The output Element

Description: This form control renders a value from the instance data, but provides no means for entering or changing data. It is used to display values from the instance, and is treated as `display:inline` for purposes of layout. Element `output` can be used to display the value at a particular location in the instance by using a binding expression; it can also be used to display the result of evaluating an XPath expression by specifying the XPath expression to be evaluated via attribute `value` instead of `ref`. Note that attributes `ref` and `value` on element `output` are mutually exclusive.

Common Attributes: Common, Single Node Binding (optional)

Special Attributes:

appearance

    This form control does not use the UI Common attribute group, but nevertheless still contains an appearance attribute, as defined above.

value

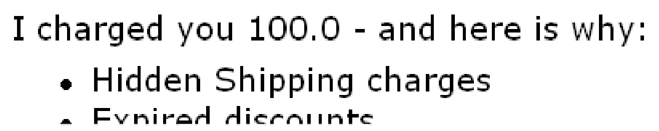
    An XPath expression to be evaluated. The result of the evaluation is rendered by the form control. If binding attributes are present to select a node, this attribute has no effect. The XPath expression is re-evaluated whenever there is a change in any node that the expression refers to.

Example:

| 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 a convenient means for display of datatypes and take into account localization and internationalization issues such as representation of numbers.

### 8.1.6 The upload Element

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.

Common Attributes: Common, UI Common, Single Node Binding

Special Attributes:

mediatype

> Space-separated list of suggested media types, used by the XForms Processor to determine the possible sources of data to upload.

incremental

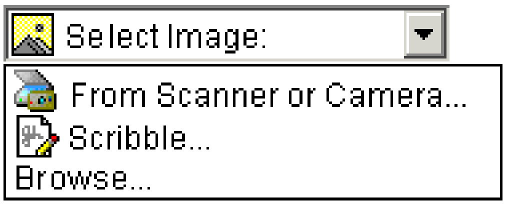
> When `true`, this form control will generate additional `xforms-value-changed` events. The default for this form control is `false`.

Example:

Example: Uploading An Image

```
<upload ref="mail/attachment" mediatype="image/*">
  <label>Select image:</label>
  <filename ref="@filename" />
  <mediatype ref="@mediatype" />
</upload>
```

A graphical browser might render this form control as follows:



Implementation Requirements:

- On activation, if child element `filename` is present and a filename is available, `upload` places the filename of the data to upload in the instance at the node indicated by the binding attributes on child element `filename`.
- On activation, if child element `mediatype` is present and a mediatype is available, `upload` places the mediatype of the data to upload in the instance at the node indicated by the binding attributes on child element `mediatype`.

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

Implementation Requirements: For base64Binary or hexBinary data binding:

- When bound to an instance data node of type `xsd:base64binary`, `xsd:hexBinary`, or a type derived by restriction thereof, on activation `upload` places the binary content in the content of the node with the indicated encoding.

Implementation Requirements: For anyURI data binding:

- When bound to an instance data node of type `xsd:anyURI` (or a type derived by restriction thereof), on activation `upload` places a URI in the content of the node.

  For security reasons, the XForms Processor must not dereference the URI bound to this form control without explicit user permission.

  **Note:**

  Implementors note that `upload` must associate the binary content, mediatype, and filename with that URI for **11.4 Serialization as multipart/related** and **11.5 Serialization as multipart/form-data** serialization.

- Implementations with a file system should support *file upload*—selecting a specific file. The types of files presented by default should reflect the mediatype specified by attribute `mediatype`, for example defaulting to only audio file types in the file dialog when the mediatype is "audio/*".

Implementation Requirements: For all data bindings:

- 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.

See the child elements `filename` **8.3.1 The filename Element** and `mediatype` **8.3.2 The mediatype Element**.

### 8.1.7 The range Element

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

Common Attributes: Common, UI Common, Single Node Binding

Special Attributes:

start

> Optional hint for the lexical starting bound for the range—a legal value for the underlying data. If provided, this value is used to further refine the constraints specified by the underlying model.

end

> Optional hint for the ending bound for the range—a legal value for the underlying data. If provided, this value is used to further refine the constraints specified by the underlying model.

step

> Optional value to use for incrementing or decrementing the value. Must be of a type capable of expressing the difference between two legal values of the underlying data.

incremental

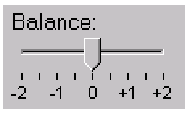
> When `true`, this form control will generate additional `xforms-value-changed` events. The default for this form control is `false`.

Example:

Example: Picking From A Range

```
<range ref="/stats/balance" start="-2.0" end="2.0" step="0.5">
  <label>Balance</label>
</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. If the instance data value is outside the upper or lower bounds, this form control must indicate an out-of-range condition. 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, in the example below, 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.*

In the event of overlapping restrictions between the underlying datatype and the `start` and `end` hints, the most restrictive range should be used.

Example:

Example: Picking a date from a range

```
<range ref="/order/shipDate" start="2001-01-01" end="2001-12-31">
  <label>Ship Date</label>
</range>
```

### 8.1.8 The trigger Element

Description: This form control is similar to the HTML element `button` and allows for user-triggered actions. This form control may also be used to construct other custom form controls.

Common Attributes: Common, UI Common, Single Node Binding (optional)

Example:

Example: Simple Trigger

```
<trigger>
  <label>Click here</label>
</trigger>
```

Data Binding Restrictions: Binds to any node. This form control does not directly interact with form data, but is affected by model item properties of the bound node, thus binding attributes are not required.

Implementation Requirements: The user agent must provide a means to generate an `DOMActivate` event on the form control. Graphical implementations might render this form control as a push-button with the label on the button face. Style sheets can be used to style this form control as an image, hyperlink, or other presentation.

### 8.1.9 The submit Element

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

Common Attributes: Common, UI Common, Single Node Binding (optional)

Special Attributes:

submission

Required reference to element `submission`.

Example:

Example: Submit

```
<submit submission="timecard">
  <label>Submit Timecard</label>
</submit>
```

Data Binding Restrictions: Binds to any node. This form control does not directly interact with form data, but is affected by model item properties of the bound node, thus binding attributes are not required.

Implementation Requirements: Upon receiving event `DOMActivate`, this form control dispatches event `xforms-submit` to the `submission` element specified by required attribute `submission`. Upon activation, this control must become unavailable for further activations until the submit process concludes with either an `xforms-submit-done` or `xforms-submit-error` event.

### 8.1.10 The select Element

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

Common Attributes: Common, UI Common, Single Node Binding

Special Attributes:

selection

> Optional attribute determining whether free entry is allowed in the list. Default is "closed".

incremental

> When `true`, this form control will generate additional `xforms-value-changed` events. The default for this form control is `true`.

Example:

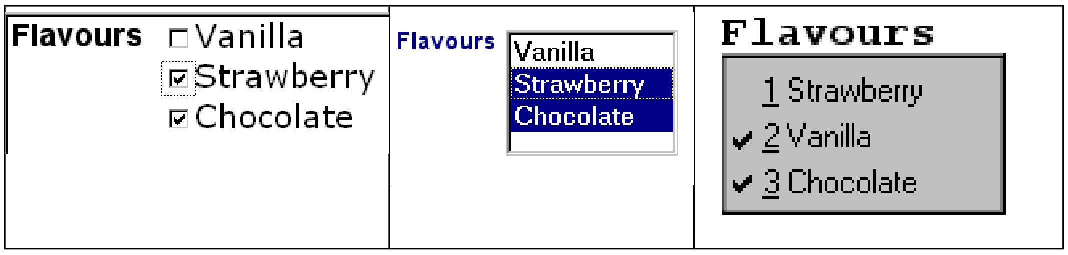
Example: Selecting Ice Cream Flavor

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

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

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

| **appearance="full"** | **appearance="compact"** | **appearance="minimal"** |
|---|---|---|



Typically, a style sheet would be used to determine the exact appearance of form controls, though a means is provided to suggest an appearance through attribute `appearance`. The value of the attribute consists of one of the following values:

> "full": all choices should be rendered at all times.
> "compact": a fixed number of choices should be rendered, with scrolling facilities as needed
> "minimal": a minimum number of choices should be rendered, with a facility to temporarily render additional choices

Data Binding Restrictions: any simpleContent capable of holding a sequence. The restriction to binding simpleContent exists when the choices are authored as part of the user interface control as shown in this section. Element `itemset` for creating dynamic selections described in **9.3.3 The itemset Element** allows the available choices to be obtained from an XForms Model, and when using that construct, the data binding restriction to simpleContent is relaxed.

**Note:**

A limitation of the XML Schema list datatypes is that white space characters in the storage values (the `value` element) are always interpreted as separators between individual data values. Therefore, authors should avoid using white space 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 Requirements: The label for each choice must be presented, allowing at any number of selections, possibly none. This form control stores the values corresponding to the selected choices as a space separated list in the location addressed by attribute `ref`. The values to be stored are either directly specified as the contents of element `value`, or specified indirectly through binding attributes on element `value`.

Note that the datatype bound to this form control may include a non-enumerated value space, e.g., `xsd:string`, or a union of a enumeration and a non-enumerated datatype (called an open enumeration). In this case, control `select` may have attribute `selection="open"`. The form control should then allow free data entry, as described in **8.1.2 The input Element**. The form control may permit multiple values to be entered through free entry.

For closed selections: If the initial instance value matches the storage value of one or more of the given items, those items are selected. If there is no match, no items are initially selected. If any selected values do not have a choice with a matching storage value, the form control must indicate an out-of-range condition.

For open selections: If the initial instance values match the storage value specified by one or more of the items, the all such matching items are selected. If the initial instance values do not match the storage value specified by one or more of the items, all such non-matching items are included as selected values, as if entered through free entry. Free entry text is handled the same as form control `input` **8.1.2 The input Element**, possibly in multiplicity. When using dynamic selections with complexTypes, open selection has no effect.

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.

### 8.1.11 The select1 Element

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

Common Attributes: Common, UI Common, Single Node Binding

Special Attributes:

selection

> Optional attribute determining whether free entry is allowed in the list. Default is "closed".

incremental

> When `true`, this form control will generate additional `xforms-value-changed` events. The default for this form control is `true`.

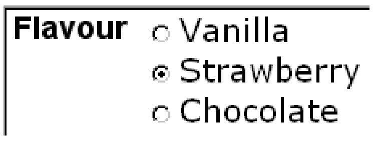
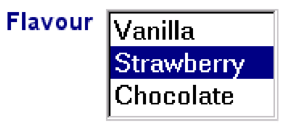
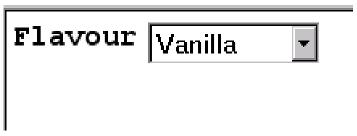
Example:

| Example: Pick A Flavor |
|---|

```
<select1 ref="my:flavor">
  <label>Flavor</label>
  <item>
    <label>Vanilla</label>
    <value>v</value>
  </item>
  <item>
    <label>Strawberry</label>
    <value>s</value>
  </item>
  <item>
    <label>Chocolate</label>
    <value>c</value>
  </item>
</select1>
```

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 instance data at the location `icecream/flavor`.

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

| appearance="full" | appearance="compact" | appearance="minimal" |
|---|---|---|
|  |  |  |

Data Binding Restrictions: Binds to any simpleContent. The restriction to binding simpleContent exists when the choices are authored as part of the user interface control as shown in this section. Element `itemset` for creating dynamic selections described in **9.3.3 The itemset Element** allows the available choices to be obtained from an XForms Model, and when using that construct, the data binding restriction to simpleContent is relaxed.

Implementation Requirements: The label 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 binding attributes on element `value`.

Note that the datatype bound to this form control may include a non-enumerated value space, e.g., `xsd:string`, or a union of a enumeration and a non-enumerated datatype (called an open enumeration). In this case, control `select1` may have attribute `selection="open"`. The form control should then allow free data entry, as described in **8.1.2 The input Element**.

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 form control must indicate an out-of-range condition..

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` **8.1.2 The input Element**.

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

## 8.2 Common Markup for Selection Controls

### 8.2.1 The choices Element

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

Common Attributes: Common

### 8.2.2 The item Element

This element specifies the storage value and label to represent an item in a list. It is found within elements `select1` and `select`, or grouped in element `choices`.

Common Attributes: Common

### 8.2.3 The value Element

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

Common Attributes: Common, Single Node Binding (optional)

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

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

## 8.3 Additional 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 `label`, the metadata can be pointed to by using a simple linking attribute `src` 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 `label` elements
- Finally, an XForms implementation could 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.

### 8.3.1 The filename Element

Binding attributes on optional element `filename` specify the location in the instance for the parent element `upload`, when activated, to place the filename for the chosen binary resource. For security reasons, `upload` must not take action due to any existing value of the node.

Common Attributes: Common, Single Node Binding

In the following example, the user is prompted to select an image. When activated, `upload` places in `mail/attachment` either the binary data of the image or a URI for it, depending on the type declared for the `mail/attachment`. The filename, perhaps "`me.jpg`", is placed in the attribute node `mail/attachment@filename`, and the mediatype, perhaps "`image/jpeg`" in the attribute node `mail/attachment@mediatype`.

Example:

```
<upload ref="mail/attachment" mediatype="image/*">
  <label>Select an image to attach</label>
  <filename ref="@filename"/>
  <mediatype ref="@mediatype"/>
</upload>
```

### 8.3.2 The mediatype Element

Binding attributes on optional element `mediatype` specify the location in the instance for the parent element `upload`, when activated, to place the mediatype of the chosen binary resource, if available.

Common Attributes: Common, Single Node Binding

### 8.3.3 The label Element

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

Common Attributes: Common, Single Node Binding (optional), Linking

Special Attributes:

Linking Attributes

Link to external label. If the link traversal fails, it is treated as an error (**4.5.3 The xforms-link-error Event**).

The label specified can exist in instance data, in a remote document, or as inline text. If more than one source of label is specified in this element, the order of precedence is: single node binding attributes, linking attributes, inline text.

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

### 8.3.4 The help Element

The optional element `help` provides a convenient way to attach help information to a form control. This is equivalent to a `<message level="modeless" ev:event="xforms-help" ev:propagate="stop">`.

Common Attributes: Common, Single Node Binding (optional), Linking

Special Attributes:

Linking Attributes

Link to external help information. If the link traversal fails, it is treated as an error (**4.5.3 The xforms-link-error Event**).

The message specified can exist in instance data, in a remote document, or as inline text. If more than one source of message is specified in this element, the order of precedence is: single node binding attributes, linking attributes, inline text.

An example of this element is at **10.1.12 The message Element**.

### 8.3.5 The hint Element

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

Common Attributes: Common, Single Node Binding (optional), Linking

Special Attributes:

Linking Attributes

Link to external hint. If the link traversal fails, it is treated as an error (**4.5.3 The xforms-link-error Event**).

The message specified can exist in instance data, in a remote document, or as inline text. If more than one source of message is specified in this element, the order of precedence is: single node binding attributes, linking attributes, inline text.

An example of this element is at **10.1.12 The message Element**.

#### 8.3.6 The alert Element

The optional element `alert` provides a convenient way to attach alert or error information to a form control. Rendering of this element is implementation-defined, and there is no default `level` such as `modal` or `ephemeral` for the displayed message.

Common Attributes: Common, Single Node Binding (optional), Linking

Special Attributes:

Linking Attributes

> Link to external alert. If the link traversal fails, it is treated as an error (**4.5.3 The xforms-link-error Event**).

The message specified can exist in instance data, in a remote document, or as inline text. If more than one source of message is specified in this element, the order of precedence is: single node binding attributes, linking attributes, inline text. See **F XForms and Styling** for examples to see how this might be presented to the user.

# 9 XForms User Interface

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

## 9.1 The XForms Group Module

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. The elements and attributes included in this module are:

| Element | Attributes | Minimal Content Model |
|---|---|---|
| group | Common, UI Common, Single Node Binding (optional) | label?, ((Form Controls)\|group\|switch\|repeat\|UI Common)* |

### 9.1.1 The group Element

The `group` element is used as a container for defining a hierarchy of form controls. Groups can be nested to create complex hierarchies. Model item properties that apply to form controls apply equally to `group`, and take precedence over model item properties applied to individual members of the `group`.

Common Attributes: Common, UI Common, Single Node Binding (optional)

**Note:**

When no model item properties apply to the binding expression on group, it can be considered as an authoring convenience for relative XPath expressions used by form controls appearing within the `group`.

When model item properties do apply, they apply to all form controls within the `group`. This means, for instance, that if a group is bound to an instance data node that is non-relevant, all child form controls will also be treated as non-relevant.

The optional `label` element has special significance when it appears as the first element child of `group`, representing a label for the entire group.

Example:

Example: Grouping Related Controls

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

Setting the input focus on a group results in the focus being set to the first form control in the navigation order within that group.

## 9.2 The XForms Switch Module

This section defines a switch construct that allows the creation of user interfaces where the user interface can be varied based on user actions and events. The elements and attributes included in this module are:

| Element | Attributes | Minimal Content Model |
|---|---|---|
| switch | Common, UI Common, Single Node Binding (optional) | case+ |
| case | Common, selected (xsd:boolean) | label?, ((Form Controls)\|group\|switch\|repeat)* |
| toggle | Common, case (xsd:IDREF) | EMPTY |

### 9.2.1 The switch Element

This element contains one or more `case` elements, any one of which is rendered at a given time.

**Note:**

This is separate from XForms `relevant` processing (see **6.1.4 The relevant Property**), which is based on the current state of the XForms Model. 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?'.

Common Attributes: Common, UI Common, Single Node Binding (optional)

Example:

Example: switch

```
<switch>
  <case id="in" selected="true">
    <input ref="yourname">
      <label>Please tell me your name</label>
      <toggle ev:event="DOMActivate" case="out"/>
    </input>
  </case>
  <case id="out" selected="false">
    <html:p>Hello <output ref="yourname" />
      <trigger id="editButton">
        <label>Edit</label>
        <toggle ev:event="DOMActivate" case="in"/>
      </trigger>
    </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 `enter` results switches to the alternate case, with a read-only `output` rendering. Activating the trigger labeled "Edit" in turn switches back to the original case.

### 9.2.2 The case Element

This element encloses markup to be conditionally rendered. The attribute `selected` determines the selected state and can be manipulated programmatically via the DOM, or declaratively via XForms Action `toggle`.

Common Attributes: Common

Special Attributes:

selected

Optional selection status for the case. The default value is "false".

If multiple `cases` 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.

### 9.2.3 The toggle Element

This XForms Action selects one possible case from an exclusive list of alternatives in a `switch`.

This action adjusts all `selected` attributes on the affected `cases` to reflect the new state, and then performs the following:

1 Dispatching an `xforms-deselect` event to the currently selected `case`.

2 Dispatching an `xform-select` event to the `case` to be selected.

Common Attributes: Common, Events

Special Attributes:

case

Required reference to a `case` section inside the conditional construct.

## 9.3 The XForms Repeat Module

The XForms specification 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. The elements and attributes included in this module are:

| Element | Attributes | Minimal Content Model |
|---|---|---|
| repeat | Common, UI Common, Node Set Binding, startindex (xsd:positiveInteger), number (xsd:nonNegativeInteger) | ((Form Controls)\|group\|repeat)* |
| itemset | Common, Node Set Binding | label, (value\|copy), (UI Common)* |
| copy | Common, Single Node Binding (optional) | EMPTY |
| insert | Common, Events, Node Set Binding, at (XPathExpression), position ("before"\|"after") | EMPTY |
| delete | Common, Events, Node Set Binding, at (XPathExpression) | EMPTY |
| setindex | Common, Events, repeat (xsd:IDREF), index (XPathExpression) | EMPTY |
| (various) | [repeat-nodeset, repeat-bind, repeat-model] (Node Set Binding attributes), repeat-startindex | N/A |

| | | |
|---|---|---|
| | (xsd:positiveInteger), repeat-number (xsd:non-NegativeInteger) | |

### 9.3.1 The repeat Element

This element defines a UI mapping over a **homogeneous collection** selected by Node Set Binding Attributes. This node-set must consist of contiguous child element nodes, with the same local name and namespace name of a common parent node. The behavior of element `repeat` with respect to non-homogeneous node-sets is undefined.

For example:

Example: Shopping Cart

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

Common Attributes: Common, UI Common, Node Set Binding

Special Attributes:

startindex

Optional 1-based initial value of the repeat index. The default value is 1.

number

Optional hint to the XForms Processor as to how many elements from the collection to display.

This element operates over a homogeneous collection by binding the encapsulated user interface controls to each element of the collection. Attributes on this element specify how many members of the collection are presented to the user at any given time. XForms Actions `insert`, `delete`, and `setindex` 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 (given 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/>
```

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">
    <label>Line Item</label>
  </input>
  <input ref="@name">
    <label>Name</label>
  </input>
</repeat>

<trigger>
  <label>Insert a new item after the current one</label>
  <action ev:event="DOMActivate">
    <insert nodeset="/my:lines/my:line" at="index('lineset')"
      position="after"/>
    <setvalue ref="/my:lines/my:line[index('lineset')]/@name"/>
    <setvalue ref="/my:lines/my:line[index('lineset')]/price">0.00</setvalue>
  </action>
</trigger>

<trigger>
  <label>remove current item</label>
  <delete ev:event="activate" nodeset="/my:lines/my:line"
    at="index('lineset')"/>
</trigger>
```

### 9.3.2 Creating Repeating Structures Via Attributes

Element `repeat` enables the creation of user interfaces for populating repeating structures. When using XForms within host languages like XHTML, it is often necessary to create repeating structures within constructs such as `table`. Thus, one might wish to use element `repeat` within a `table` to create the rows of a table, where each row of the table binds to a distinct member of a homogeneous collection. Since `html:table` doesn't (and probably never will) allow `xforms:repeat` elements as children, another syntax is needed.

Example: Tables And Repeating Structures

```
<table>
  <repeat nodeset="...">
    <tr>
      <td>...</td>
      ...
    </tr>
  </repeat>
</table>
```

More generally, there is a need to integrate repeat behavior into host languages at points where the content model of the host language does not or cannot provide the appropriate extension hooks via modularization. To accommodate this, XForms 1.0 defines an alternative syntax that is functionally equivalent to the `repeat` element, using the following attributes:

```
repeat-model
repeat-bind
repeat-nodeset
repeat-startindex
repeat-number
```

The above attributes are equivalent to the `repeat` attributes of the same name, but without the prefix `repeat-`. A host language can include these attributes in the appropriate places to enable repeating constructs. For example, a version of XHTML might use:

Example: Tables And Repeating Structures

```
<html:table xforms:repeat-nodeset="...">
  <html:tr>
    <html:td><xforms:output ref="..."/></html:td>
  </html:tr>
</html:table>
```

Which could be validated against an appropriately configured XHTML Schema that includes the XForms Repeat module. Note that what gets repeated is the child elements of the element with the `repeat-` attributes.

This should be thought purely as a syntactic transformation, i.e., there is no change to repeat processing semantics. Further, for purposes of understanding the above as a pure syntactic transformation, element `repeat` can be viewed as containing an anonymous `group` that wraps the contents of element `repeat`. Thus, consider the following:

```
<repeat ...>
  ...
</repeat>
```

is equivalent to

```
<repeat ...>
  <group>...</group>
</repeat>
```

Which is equivalent to

```
<group repeat-...>
  ...
</group>
```

Additionally, when using XForms Action `setindex`, attribute `repeat` of type `idref` can point to any element carrying the repeat attributes. Similarly, when using function `index` against a repeating structure created via the `repeat`-attributes, the `id` of that element can be used as the argument to function `index`.

### 9.3.3 The itemset Element

This element allows the creation of dynamic selections within controls `select` and `select1`, where the available choices are determined at run-time. The node-set that holds the available choices is specified via attribute `nodeset`. As with `repeat`, this nodeset should refer to a homogeneous collection. Child elements `label` and `value` indirectly specify the label and storage values. Notice that the run-time effect of `itemset` is the same as using element `choices` to statically author the available choices.

Common Attributes: Common, Node Set Binding

> **Note:**
>
> Whenever a `refresh` event is dispatched the `nodeset` is re-evaluated to update the list of available choices.

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

Example: Dynamic Choice Of Ice Cream Flavors

```
<model id="cone">
  <instance>
    <my:icecream>
      <my:order/>
    </my:icecream>
  </instance>
</model>
<model id="flavors">
  <instance>
    <my:flavors>
      <my:flavor type="v">
        <my:description>Vanilla</my:description>
      </my:flavor>
      <my:flavor type="s">
        <my:description>Strawberry</my:description>
      </my:flavor>
      <my:flavor type="c">
        <my:description>Chocolate</my:description>
      </my:flavor>
    </my:flavors>
  </instance>
</model>
<!-- user interaction markup -->
<select model="cone" ref="my:order">
  <label>Flavors</label>
  <itemset model="flavors" nodeset="/my:flavors/my:flavor">
    <label ref="my:description"/>
    <copy ref="my:description"/>
  </itemset>
</select>

<!-- For all three items selected, this example produces instance data like
     <my:icecream>
       <my:order>
         <my:description>Vanilla</my:description>
         <my:description>Strawberry</my:description>
         <my:description>Chocolate</my:description>
       </my:order>
     </my:icecream>
-->
```

### 9.3.4 The copy Element

Structurally, this element is similar to **8.2.3 The value Element**. It differs in that it can only be used within `itemset`, and that it works with subtrees of instance data rather than simple values.

Common Attributes: Common, Single Node Binding (optional)

When an `item` becomes selected, the following rules apply:

- The target node, selected by the binding attributes on the list form control, must be an element node, otherwise an exception results (**4.5.1 The xforms-binding-exception Event**).
- The element node associated with the `item`, selected by the binding attributes on `copy`, is deep copied as a child of the target node.
- A full computational dependency rebuild is done.

When an `item` becomes unselected, the following rules apply:

- The target node, selected by the binding attributes on the list form control, must be an element node, otherwise an exception results (**4.5.1 The xforms-binding-exception Event**).
- The child element node associated with the `item`, selected by the binding attributes on `copy`, is deleted.
- A full computational dependency rebuild.

### 9.3.5 The insert Element

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 location 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 modified to remain as unique values in the instance data.

Common Attributes: Common, Events, Node Set Binding

Special Attributes:

at

  Required XPath expression evaluated to determine insert location.

position

  Required selector ("before" or "after") of insert before/after behavior.

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 location 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:

a 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`.

b If the result is `NaN`, the insert appends to the end of the node-set.

c If the resulting index is outside the valid range of the node-set, it is replaced with either `1` or the size of the node-set, whichever is closer.

3 The index for any repeating sequence that is bound to the homogeneous collection where the node was added is updated to point to the newly added node. The indexes for inner nested repeat collections are re-initialized to 1.

4 If the insert is successful, the event `xforms-insert` is dispatched.

This action results in the insertion of newly created data nodes into the XForms instance data. Such nodes are constructed as defined in the initialization section of the processing model—see **4.2 Initialization Events**. 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.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

An example of using `insert` with a repeating structure is located at **9.3.1 The repeat Element**. Note that XForms Action `setvalue` can be used in conjunction with `insert` to provide initial values for the newly inserted nodes.

### 9.3.6 The delete Element

This action deletes nodes from the instance data.

Common Attributes: Common, Events, Node Set Binding

Special Attributes:

at

Required XPath expression evaluated to determine delete location.

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 empty, the delete action has no effect.

2 The `n`-th element node is deleted from the instance data, where `n` represents the number returned from node-set index evaluation, defined in **9.3.5 The insert Element**. If no `n`th node exists, the operation has no effect.

3 The index should point to the same node after a delete as it did before the delete except:

> When the last remaining item in the collection is removed, the index position becomes 0.
>
> When the index was pointing to the deleted node, which was the last item in the collection, the index will point to the new last node of the collection and the index of inner repeats is reinitialized.
>
> When the index was pointing to the deleted node, which was not the last item in the collection, the index position is not changed and the index of inner repeats is re-initialized.

To re-initialize a repeat means to change the index to 0 if it is empty, otherwise 1.

4 If the delete is successful, the event `xforms-delete` is dispatched.

This action results in deletion of nodes in the instance data.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

An example of using `delete` with a repeating structure is located at **9.3.1 The repeat Element**.

### 9.3.7 The setindex Element

This action marks a specific item as current in a repeating sequence (within **9.3.1 The repeat Element**).

Common Attributes: Common, Events

Special Attributes:

repeat

Required reference to a repeating element.

index

Required XPath expression that evaluates to a 1-based offset into the sequence.

If the selected index is 0 or less, an `xforms-scroll-first` event is dispatched and the index is set to 1. If the selected index is greater than the index of the last repeat item, an `xforms-scroll-last` event is dispatched and the index is set to that of the last item. The indexes for inner nested repeat collections are re-initialized to 1. The implementation data structures for tracking computational dependencies are rebuilt or updated as a result of this action.

### 9.3.8 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 **4.2.2 The xforms-model-construct-done Event**), 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 *index* for this repeating structure is initialized to the value of `startindex`.

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`.

The processing model for repeating structures uses an *index* that points to the *current* item in the instance data. This repeat index is accessed via XForms function `index` **7.8.5 The index() Function** and manipulated via XForms Action `setindex` **9.3.7 The setindex Element**. This index 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 index. Repeat processing uses XPath expressions to address the collection over which element `repeat` operates. The initial instance data supplies the prototypical member of the homogeneous collection, and this is used during UI initialization—**4.2.2 The xforms-model-construct-done Event**—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.

### 9.3.9 Nested Repeats

It is possible to nest repeat elements to create more powerful user interface for editing structured data. **G.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 index always starts from `1`. Consider the following `insert` statement that appears as part of that example.

Example: Repeat Index and Nested Repeats

```
<xforms:insert nodeset="/bookmarks/section[index('repeatSections')]/bookmark"
               at="index('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 index—as returned by XForms function `index`—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.

### 9.3.10 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 index should be made available to the user 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

This chapter defines an XML Events-based [XML Events] common set of actions that can be invoked in response to events.

**Note:**

XForms itself defines no method for script-based event handling. The definition of such facilities is a responsibility of the hosting language.

## 10.1 The XForms Action Module

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.

The elements and attributes included in this module are:

| Element | Attributes | Minimal Content Model |
|---|---|---|
| action | Common, Events | (Action)+ |
| dispatch | Common, Events, name (xsd:NMTOKEN), target (xsd:IDREF), bubbles (xsd:boolean), cancelable (xsd:boolean) | EMPTY |
| rebuild | Common, Events, model (xsd:IDREF) | EMPTY |
| recalculate | Common, Events, model (xsd:IDREF) | EMPTY |
| revalidate | Common, Events, model (xsd:IDREF) | EMPTY |
| refresh | Common, Events, model (xsd:IDREF) | EMPTY |
| setfocus | Common, Events, control (xsd:IDREF) | EMPTY |
| load | Common, Events, Single Node Binding (optional), resource (xsd:anyURI), show ("new" \| "replace") | EMPTY |
| setvalue | Common, Events, Single Node Binding, value (XPathExpression) | PCDATA |
| send | Common, Events, submission (xsd:IDREF) | EMPTY |
| reset | Common, Events, model (xsd:IDREF) | EMPTY |
| message | Common, Events, Single Node Binding (optional), Linking, level ("ephemeral" \| "modeless" \| "modal") | (PCDATA\|UI Inline)* |

See also: **9.2.3 The toggle Element**; **9.3.5 The insert Element**; **9.3.6 The delete Element**; and **9.3.7 The setindex Element**.

This module also defines the content set "Action", which includes the following elements (of these, `toggle` comes from the XForms Switch module, and `insert`, `delete`, and `setindex` come from the XForms Repeat module):

```
(action|dispatch|rebuild|refresh|recalculate|revalidate|setfocus|
 load|setvalue|send|reset|message|toggle|insert|delete|setindex)*
```

Additionally, this module defines the attribute group "XML Events", which includes all of the "global" attributes defined in that specification ([XML Events]).

The following example shows how events can be used:

Example: Action Syntax

```
<xforms:trigger>
  <xforms:label>Reset</xforms:label>
  <xforms:reset ev:event="DOMActivate" model="thismodel"/>
</xforms:trigger>
```

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

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

Name
Common Attributes
Special Attributes
Description of behavior

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].

### 10.1.1 The action Element

Action `action` is used to group multiple actions.

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

Common Attributes: Common, Events

Example: Grouping Actions

```
<trigger>
  <label>Click me</label>
  <action ev:event="DOMActivate">
    <reset model="thismodel"/>
    <setvalue ref="."/>
  </action>
</trigger>
```

Notice that in the above example, `ev:event="DOMActivate"` occurs on element `action`. Placing `ev:event="DOMActivate"` on either or both of the contained actions will have no effect. This is because the above example relies on the defaulting of [XML Events] 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="DOMActivateD"` on the children of element `action` therefore causes element `action` to become the *observer* for the individual events. Consequently, these actions will never be triggered since events arrive at element `trigger`, not element `action`.

**Deferred Updates**: Many XForms Actions have a deferred effect on the instance data when specified as a descendant of an `action` element.

Implementations are free to use any strategy to accomplish deferred updates, but the end result must be as follows: Instance data changes performed by a set of actions do not result in immediate computation dependency rebuilding, recalculation, revalidate and form control refreshing until the termination of the outermost action handler, as described here. Each outermost action handler can be thought of as having a set of Boolean flags, initially `false`, to indicate whether each of the actions `rebuild`, `recalculate`, `revalidate`, and `refresh` are required upon termination of the outer action handler.

Actions that directly invoke rebuild, recalculate, revalidate, or refresh always have an immediate effect, and clear the corresponding flag. The XForms Actions in this category are:

```
rebuild
recalculate
revalidate
refresh
```

XForms Actions that change the tree structure of instance data result in setting all four flags to `true`. The XForms Actions in this category are:

```
insert
delete
```

XForms Actions that change only the value of an instance node results in setting the flags for `recalculate`, `revalidate`, and `refresh` to `true` and making no change to the flag for `rebuild`. The XForms Actions in this category are:

setvalue

Finally, the `reset` action takes effect immediately and clears all of the flags.

### 10.1.2 The dispatch Element

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

1 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 **4 Processing Model** apply.

2 An event created by the XForms author with no predefined XForms semantics and as such not handled by default by the XForms Processor.

Common Attributes: Common, Events

Special Attributes:

name

Required name of the event to dispatch.

target

Required reference to the event target.

bubbles

Optional boolean indicating if this event bubbles—as defined in [DOM2 Events]. The default value depends on the definition of a custom event. For predefined events, this attribute has no effect.

cancelable

Optional boolean indicating if this event is cancelable—as defined in [DOM2 Events]. The default value depends on the definition of a custom event. For predefined events, this attribute has no effect.

### 10.1.3 The rebuild Element

This action causes the processing of `xforms-rebuild` to happen, bypassing the normal event flow. This action results in the XForms Processor rebuilding any internal data structures used to track computational dependencies among instance data nodes —see **4.3.7 The xforms-rebuild Event**.

Common Attributes: Common, Events

Special Attributes:

model

Required IDREF of the `model` to be processed.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

### 10.1.4 The recalculate Element

This action causes the processing of `xforms-recalculate` to happen, bypassing the normal event flow. As a result, instance data nodes whose values need to be recalculated are updated as specified in the processing model—see **4.3.6 The xforms-recalculate Event**.

Common Attributes: Common, Events

Special Attributes:

model

Required IDREF of the `model` to be processed.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

### 10.1.5 The revalidate Element

This action causes the processing of `xforms-revalidate` to happen, bypassing the normal event flow. This results in the instance data being revalidated as specified by the processing model—see **4.3.5 The xforms-revalidate Event**.

Common Attributes: Common, Events

Special Attributes:

model

Required IDREF of the `model` to be processed.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

### 10.1.6 The refresh Element

This action causes the processing of `xforms-refresh` to happen, bypassing the normal event flow. 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 **4.3.4 The xforms-refresh Event**.

Common Attributes: Common, Events

Special Attributes:

model

Required IDREF of the `model` to be processed.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

### 10.1.7 The setfocus Element

This action sets focus to the form control identified by attribute `control` by dispatching an `xforms-focus` event (**4.3.2 The xforms-focus Event**). Note that this event is implicitly invoked to implement XForms accessibility features such as `accesskey`.

Common Attributes: Common, Events

Special Attributes:

control

Required reference to a form control.

Setting focus to a repeating structure sets the focus to the repeat item represented by the repeat index.

### 10.1.8 The load Element

This action traverses the specified link.

Common Attributes: Common, Events, Single Node Binding (optional)

Special Attributes:

resource

Link to external resource to load, defined as an [XLink 1.0] link between this element and the remote resource indicated. No XLink `actuate` value is defined, since control of actuation is defined by XML Events. The XLink `show` value depends on the `show` attribute. If the link traversal fails, it is treated as an error (**4.5.3 The xforms-link-error Event**).

show

Optional link behavior specifier.

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

Possible values for attribute `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 presentation context, e.g., a new window. 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.

### 10.1.9 The setvalue Element

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

Common Attributes: Common, Events, Single Node Binding

Special Attributes:

value

> Optional 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 (""). If both are present, the `value` attribute is used.

All strings are inserted into the instance data as follows:

- Element nodes: If the element has any child text nodes, the first text node is replaced with one corresponding to the new value. If no child text nodes are present, a text node is created, corresponding to the new value, and appended as the first child node.
- Attribute nodes: The string-value of the attribute is replaced with a string corresponding to the new value.
- Text nodes: The text node is replaced with a new one corresponding to the new value.
- Namespace, processing instruction, comment, and the XPath root node: behavior is undefined.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

### 10.1.10 The send Element

This action initiates submit processing by dispatching an `xforms-submit` event. Processing of event `xforms-submit` is defined in the processing model—see **4.3.9 The xforms-submit Event**.

Common Attributes: Common, Events

Special Attributes:

submission

Required reference to a `submission` element.

**Note:**

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

```
<dispatch target="mysubmitinfo" name="xforms-submit"/>
```

### 10.1.11 The reset Element

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 **4.3.8 The xforms-reset Event**.

Common Attributes: Common, Events

Special Attributes:

model

Required selection of instance data for reset, defined in **3.2.3 Single-Node Binding Attributes**.

**Note:**

If this action is contained within an `action` element, it has special deferred update behavior (**10.1.1 The action Element**).

#### 10.1.12 The message Element

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

Common Attributes: Common, Events, Single Node Binding (optional)

Special Attributes:

Linking Attributes

Link to external message. If the link traversal fails, it is treated as an error (**4.5.3 The xforms-link-error Event**).

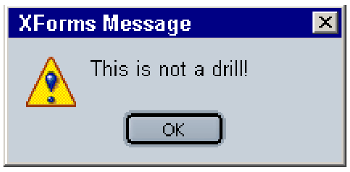
level

Required message level identifier, one of ("ephemeral"|"modeless"|"modal"|QName-but-not-NCName). This specification does not define behavior for QName values.

The message specified can exist in instance data, in a remote document, or as inline text. If more than one source of message is specified in this element, the order of precedence is: single node binding attributes, linking attributes, inline text.

A graphical browser might render a modal message as follows:

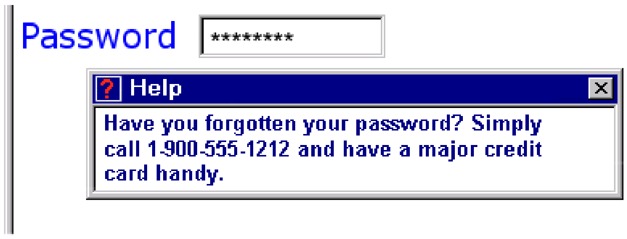
```
<model>
  <message level="modal" ev:event="xforms-ready">This is not a drill!</message>
  ...
</model>
```


XForms Message

This is not a drill!

A modeless message is the foundation for displaying a `help` message, which a graphical browser might render as follows:

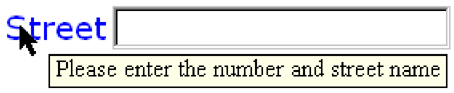
```
<secret ref="/login/password">
  <label>Password</label>
  <help>Have you forgotten your password? Simply call 1-900-555-1212 and have
        a major credit card handy.</help>
</secret>
```



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

An ephemeral message is the foundation for displaying a `hint` message, which a graphical browser might render as follows:

```
<input ref="po/address/street1">
  <label>Street</label>
  <hint>Please enter the number and street name</hint>
</input>
```



Please enter the number and street name

### 10.1.13 Actions `insert`, `delete` and `setindex`

In addition to the action handlers detailed in this chapter, XForms defines three actions as part of the XForms Repeat module: **9.3.5 The insert Element**, **9.3.6 The delete Element**, and **9.3.7 The setindex Element**.

# 11 Submit

XForms is designed to gather instance data, serialize it into an external representation, and submit it with a protocol. XForms defines a set of options for serialization and submission. The following sections define the processing of instance data for submission, and the behavior for the serialization and submission options.

## 11.1 The xforms-submit Event

Submission begins with the default action for a `xforms-submit` event.

Target: `submission`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Under no circumstances may more than a single concurrent submit process be under way for a particular XForms Model. From the start of the default action of `xforms-submit`, until the completion of the default action for `xforms-submit-done` or `xforms-submit-error`, the default action for subsequent `xforms-submit` events is to do nothing.

Otherwise, default action for this event results in the following steps:

1 A node from the instance data is selected, based on attributes on the `submission` element. The indicated node and all nodes for which it is an ancestor are considered for the remainder of the submit process.

2 All selected instance data is revalidated, according to the rules at **4.3.5 The xforms-revalidate Event**, taking into account only namespace nodes considered for serialization as described at **3.3.3 The submission Element**.. Any invalid instance data stops submit processing after dispatching event `xforms-submit-error`.

3 Selected instance data is serialized according to the rules stated at **11.2 Submission Options**.

4 Serialized instance data is submitted using the protocol indicated by the rules stated at **11.2 Submission Options**.

5 The response returned from the submission is applied as follows:

For a success response including a body, when the value of the `replace` attribute on element `submission` is "`all`", the event `xforms-submit-done` is dispatched, and submit processing concludes with entire containing document being replaced with the returned body.

For a success response including a body of an XML media type, when the value of the `replace` attribute on element `submission` is "`instance`", the response is parsed as XML and all of the internal instance data corresponding to the submitted instance is replaced with the result, using the same processing as remote instance data retrieved through `src`, and the `xforms-model-construct` event is dispatched to element `model`. Submit processing then concludes after dispatching `xforms-submit-done`.

For a success response including a body of a non-XML media type, when the value of the `replace` attribute on element `submission` is "`instance`", nothing in the document is replaced and submit processing concludes after dispatching `xforms-submit-error`.

For a success response including a body, when the value of the `replace` attribute on element `submission` is "`none`", submit processing concludes after dispatching `xforms-submit-done`.

For a success response not including a body, submit processing concludes after dispatching `xforms-submit-done`.

Behaviors of other possible values for attribute `replace` are not defined in this specification.

For an error response nothing in the document is replaced, and submit processing concludes after dispatching `xforms-submit-error`.

## 11.2 Submission Options

The XForms Model specifies a `submission` element containing the following attributes that affect serialization and submission. This section summarizes the behaviors for the allowable values of these attributes, and introduces the following sections that define the behavior for serialization and submission. (See **3.3.3 The submission Element** for additional `submission` attributes that affect serialization.)

- `action` (xsd:anyURI)
- `method` (xsd:string, enumerated below)

For the URI scheme of `action`, XForms normatively defines a binding to HTTP/1.1 [RFC 2616].

**Note:**

Other bindings, in particular to the URI scheme "mailto:" may, and the schemes "https:" and "file:" should, be supported. Bindings to these schemes are not normatively defined in XForms. Implementations that choose to provide a binding to these schemes should pay particular attention to privacy and security concerns. Within the "http:" and "https:" schemes, form creators are encouraged to follow the finding of the W3C Technical Architecture Group on when to use the GET method: [TAG Finding 7]

The `method` attribute determines the serialization format, and the URI scheme used in the `action` attribute determines the submit protocol, according to the following table:

| URI scheme | `method` | Serialization | Submission |
|---|---|---|---|
| http https mailto | "post" | `application/xml` | HTTP POST or equivalent |
| http https file | "get" | `application/x-www-form-urlencoded` | HTTP GET or equivalent |
| http https file | "put" | `application/xml` | HTTP PUT or equivalent |
| http https mailto | "multipart-post" | `multipart/related` | HTTP POST or equivalent |

| URI scheme | method | Serialization | Submission |
|---|---|---|---|
| http https mailto | "form-data-post" | `multipart/form-data` | HTTP POST or equivalent |
| http https mailto | "urlencoded-post" (Deprecated) | `application/x-www-form-urlencoded` | HTTP POST or equivalent |
| (any) | any other QNAME with no prefix | N/A | N/A |
| (any) | any QNAME with a prefix | implementation-defined | implementation-defined |

**Note:**

Foreign-namespaced attributes are allowed on element `submission`, but no behavior is defined by XForms 1.0.

## 11.3 Serialization as application/xml

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 capable of submission of binary content.

The steps for serialization are as follows:

1 An XML document is produced following the rules of the XML output method defined in [XSLT 1.0] section 16 and 16.1, using the values supplied as attributes of the `submission` element.

a Handling of namespace nodes: The default behavior is that every namespace node is serialized according to the rules of the XML output method, so that at least one namespace declaration appears in the serialized XML for each in-scope namespace. Additional inherited namespaces are declared on the root element of the serialized XML. If, however, attribute `includenamespaceprefixes` on element `submission` is present, then all namespace declarations not visibly utilized in the instance data (as defined in [Exc-C14N]) and the default namespace if it is empty are excluded from the root element serialization, unless the corresponding namespace prefix is listed in the `includenamespaceprefixes` attribute. The special value `#default` represents the default namespace.

b Mediatype: By default, the mediatype of the serialized XML instance is `application/xml`, but can be changed to a compatible type using element `submission` attribute `mediatype`. Authors should ensure that the type specified is compatible with `application/xml`.

## 11.4 Serialization as multipart/related

This format is intended for integration of XForms into environments that involve large amounts of binary data where the inclusion of the data as `xsd:base64Binary` or `xsd:hexBinary` is undesirable.

In this format, XML instance data is serialized as one part of the [RFC 2387] `multipart/related` message, using the rules as described in **11.3 Serialization as application/xml**. Binary content from `xsd:anyURI` instance nodes populated by the `upload` (see **8.1.6 The upload Element**) control is serialized in separate parts of the [RFC 2387] `multipart/related` message.

This format follows the rules of `multipart/related` MIME data streams for in [RFC 2387], with specific requirements of this serialization listed below:

- `multipart/related` message header requirements:

  Must contain a `type` parameter of the mediatype of the serialized XML instance.

  Must contain a `start` parameter referring to the Content-ID first body part (root).

- First body part (root) requirements:

  Must have `Content-Type` parameter of the type specified by the `submission mediatype` attribute.

  Content is serialized by the rules at **11.3 Serialization as application/xml**.

- Subsequent part requirements:

  One part for each node with a datatype of `xsd:anyURI` populated by `upload` with:

  A `Content-Type` header that represents the type of the attachment if known, otherwise `application/octet-stream`.

  A `Content-Transfer-Encoding` header.

  A `Content-ID` header whose value matches the URI in the associated instance data node.

  The binary content associated with the URI, serialized according to the `Content-Transfer-Encoding` heading.

Example: multipart/related

```
Content-Type: multipart/related; boundary=f93dcbA3; type=application/xml; start="<98011
Content-Length: xxx
--f93dcbA3
Content-Type: application/xml; charset=UTF-8
Content-ID: <980119.X53GGT@example.com>
<?xml version="1.0"?>
<uploadDocument>
  <title>My Proposal</title>
  <author>E. X. Ample</author>
  <summary>A proposal for a new project.</summary>
  <notes image="cid:980119.X17AXM@example.com">(see handwritten region)</notes>
  <keywords>project proposal funding</keywords>
  <readonly>false</readonly>
  <filename>image.png</filename>
  <content>cid:980119.X25MNC@example.com</content>
</uploadDocument>
--f93dcbA3
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <980119.X25MNC@example.com>
...Binary data here...
--f93dcbA3
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <980119.X17AXM@example.com>
...Binary data here...
--f93dcbA3--
```

## 11.5 Serialization as multipart/form-data

This format is for legacy compatibility to permit the use of XForms clients with [RFC 2388] servers. This method is suitable for the persistence of binary content. Contextual path information, attribute values, namespaces and namespace prefixes are not preserved. As a result, different elements might serialize to the same name.

> **Note:**
>
> Existing HTML user agents fail to encode special characters (such as double quotes) and non-ASCII characters in the `Content-Disposition: form-data` `name` and `filename` parameters. Since this serialization method is supported for legacy applications only, new applications should use `application/xml` or `multipart/related`.

This format follows the rules for `multipart/form-data` MIME data streams in [RFC 2388], with specific requirements of this serialization listed below:

- Each element node is visited in document order.
- Each element that has exactly one text node child is selected for inclusion.

- Element nodes selected for inclusion are as encoded as `Content-Disposition: form-data` MIME parts as defined in [RFC 2387], with the `name` parameter being the element local name.
- Element nodes of any datatype populated by `upload` are serialized as the specified content and additionally have a `Content-Disposition filename` parameter, if available.
- The `Content-Type` must be `text/plain` except for `xsd:base64Binary`, `xsd:hexBinary`, and derived types, in which case the header represents the media type of the attachment if known, otherwise `application/octet-stream`. If a character set is applicable, the `Content-Type` may have a `charset` parameter.

Example:

Example: multipart/form-data

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

--AaB03x
Content-Disposition: form-data; name="document"; filename="b.txt"
Content-Type: text/plain; charset=iso-8859-1
This is a file.
It has two lines.
--AaB03x
Content-Disposition: form-data; name="title"
Content-Type: text/plain; charset=iso-8859-1
A File
--AaB03x
Content-Disposition: form-data; name="summary"
Content-Type: text/plain; charset=iso-8859-1
This is my file
file test
--AaB03x--
```

## 11.6 Serialization as application/x-www-form-urlencoded

This serialization format is designed to allow the use of a form to gather the data necessary to produce a URI that names a resource and for accessing that resource with an HTTP GET operation.

This format represents an extension of the [XHTML 1.0] form content type `application/x-www-form-urlencoded` with specific rules for encoding non-ASCII and reserved characters.

This format is not suitable for the persistence of binary content. Therefore, it is recommended that forms capable of containing binary content use another serialization method.

The steps for serialization are as follows:

1 Each element node is visited in document order. Each element that has one text node child is selected for inclusion. Note that attribute information is not preserved.

2 Element nodes selected for inclusion are encoded as `EltName=value{sep}`, where `=` is a literal character, `{sep}` is the separator character from the `separator` attribute on `submission`, `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 namespaces or namespace prefixes. As a result, different elements might serialize to the same name.

   The encoding of `EltName` and `value` are as follows: space characters are replaced by `+`, and then non-ASCII and reserved characters (as defined by [RFC 2396] as amended by subsequent documents in the IETF track) are escaped by replacing the character with one or more octets of the UTF-8 representation of the character, with each octet in turn replaced by `%HH`, where `HH` represents the uppercase hexadecimal notation for the octet value and `%` is a literal character. Line breaks are represented as "CR LF" pairs (i.e., `%0D%0A`).

3 All such encodings are concatenated, maintaining document order.

Example:

Example: application/x-www-form-urlencoded

```
GivenName=Ren%C3%A9;
```

This format consists of simple name-value pairs.

```
<PersonName title="Mr">
  <GivenName>René</GivenName>
</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.

## 11.7 The post, multipart-post, form-data-post, and urlencoded-post Submit Methods

These submit methods represent HTTP POST or the equivalent concept (such as a mail message). The serialized form data is delivered as the message body.

## 11.8 The put Submit Method

This submit method represents HTTP PUT or the equivalent concept (such as writing to a local file). The serialized form data is delivered as the message body.

## 11.9 The get Submit Method

This submit method represents HTTP GET or the equivalent concept. The serialized form data is delivered as part of the URI that is requested during the submit process.

This method is not suitable for submission of forms that are intended to change state or cause other actions to take place at the server. See [RFC 2616] for recommended uses of HTTP GET.

The URI is constructed as follows:

- The submit URI from the `action` attribute is examined. If it does not already contain a `?` (question mark) character, one is appended. If it does already contain a question mark character, then a separator character from the attribute `separator` is appended.
- The serialized form data is appended to the URI.

No message body is sent with the request.

# 12 Conformance

## 12.1 Conformance Levels

The XForms specification is intended for implementation on hardware platforms of all sizes, from tiny hand-held devices to high-powered servers. For this reason, a separate document is being developed to describe a conformance profile of XForms that can be processed with fewer resources.

### 12.1.1 XForms Full

This conformance level is suitable for more powerful forms processing, such as might be found on a standard desktop browser or a distributed XForms Processor involving server-side components. **XForms Full** implementations must return "`full`" from the `property` method (defined at **7.9.1 The property() Function** ) called with the "`conformance-level`" parameter string.

## 12.2 Conformance Description

### 12.2.1 Conforming XForms Processors

All XForms Processors must conform to the following specifications, except as qualified below:

- [XML 1.0]
- [XML Names]
- [XML Events]
- [XPath 1.0] Implementing all features.
- [XML Schema part 2]

All XForms Processors must fully support the following XForms modules: Core; MustUnderstand; Form Controls; Group; Switch; Repeat; and Action.

All XForms Processors must also support: the XForms Processing Model and all events listed at **4 Processing Model**; the `http` scheme for processing xsd:anyURI; all serialization methods defined at **11 Submit**.

A host language may introduce additional conformance requirements.

XForms Full Processors must implement all required features defined in this specification.

### 12.2.2 Conforming XForms Documents

All XForms Containing Documents must conform to the following specifications, except as qualified below:

- [XML 1.0]
- [XML Names]
- [XML Events]
- [XML Schema part 2]

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, `group`, `repeat`, or `switch`. Individual XForms fragments must be schema-valid against the Schema for XForms (**A Schema for XForms**).

A host language may introduce additional conformance requirements.

All XForms Full conforming documents must conform to all required portions of this specification.

### 12.2.3 Conforming XForms Generators

XForms generators must generate conforming XForms documents.

# 13 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 locater.]

Binding expression

> [Definition: An [XPath 1.0] PathExpr used in a binding.]

Model Binding expression

> [Definition: An [XPath 1.0] PathExpr used in a binding that declares a model item property.]

UI or Action Binding expression

> [Definition: An [XPath 1.0] PathExpr used in binding a form control to the instance, or to specify the node or node-set for operation by an action.]

Computed expression

> [Definition: An [XPath 1.0] 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.]

First node rule

> [Definition: When a UI Single-Node Binding attribute selects a node-set of size > 1, the first node in the node-set is used.]

Form control

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

Host language

[Definition: An XML vocabulary, such as XHTML, into which XForms is embedded.]

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 1.0] 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 property

[Definition: An XForms-specific annotation to an instance data node.]

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 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

The normative XML Schema for XForms is located at http://www.w3.org/MarkUp/Forms/2002/XForms-Schema.xsd.

## A.1 Schema for XML Events

This XML Schema for XML Events is referenced by the XML Schema for XForms, and located at http://www.w3.org/TR/2003/REC-xml-events-20031014/#a_schema_attribs.

# B References

## B.1 Normative References

Exc-C14N
*Exclusive XML Canonicalization Version 1.0*, J. Boyer, D. Eastlake 3rd, J. Reagle, 2002. W3C Recommendation available at http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/.

RFC 2119
*RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, 1997. Available at http://www.ietf.org/rfc/rfc2119.txt.

RFC 2387
*RFC 2387: The MIME Multipart/Related Content-type*, E. Levinson, 1998. Available at: http://www.ietf.org/rfc/rfc2387.txt.

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

RFC 2396
*RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, 1998. Available at: http://www.ietf.org/rfc/rfc2396.txt.

RFC 2616
*RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee,1999. Available at: http://www.ietf.org/rfc/rfc2616.txt.

XHTML Modularization
*Modularization of XHTML*, M. Altheim, et al., 2001. W3C Recommendation available at http://www.w3.org/TR/xhtml-modularization/.

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 Base
*XML Base*, Jonathan Marsh, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlbase/.

XML Events

*XML Events - An events syntax for XML*, Steven Pemberton, T. V. Raman, Shane P. McCarron, 2002. W3C Proposed Recommendation available at: http://www.w3.org/TR/xml-events/.

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

## B.2 Informative References

Algorithms
: *The Art of Computer Programming: Volume 1 Fundamental Algorithms*, D. E. Knuth, Addison-Wesley, Reading, MA. 1968. Third edition, 1997. ISBN:0-2018-9683-4.

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.

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

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

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/.

EXSLT
*EXSLT Web site*. Available at http://www.exslt.org.

Java Unicode Blocks
*Java 2 Platform, Standard Edition, v 1.4.0 API Specification; Class Character.UnicodeBlock*, Sun Microsystems, Inc, 2002. Available at http://java.sun.com/j2se/1.4/docs/api/java/lang/Character.UnicodeBlock.html.

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/.

SVG 1.1
*SVG 1.1*, Jon Ferraiolo, FUJISAWA Jun, Dean Jackson, 2003. W3C Recommendation available at: http://www.w3.org/TR/SVG11/.

TAG Finding 7
*TAG Finding: URIs, Addressability, and the use of HTTP GET*, Dan Connolly, 2002. Available at: http://www.w3.org/2001/tag/doc/get7

UAAG 1.0
*User Agent Accessibility Guidelines 1.0*, Ian Jacobs, Jon Gunderson, Eric Hansen, 2002. Working Draft available at http://www.w3.org/TR/UAAG10/.

Unicode Scripts
*Script Names*, Mark Davis, 2001. Unicode Technical Report #24 available at http://www.unicode.org/unicode/reports/tr24/.

XForms Basic
*XForms Basic Profile*, Micah Dubinko, T. V. Raman, 2003. W3C Candidate Recommendation available at: http://www.w3.org/TR/xforms-basic/.

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.

XLink 1.0
*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 Schema part 0
*XML Schema Part 0: Primer*, David C. Fallside, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-0/.

# C Privacy Considerations

## C.1 Using P3P with XForms

P3P privacy policies may be associated with any forms transmitted over HTTP that have URIs associated with them. In the future, mechanisms may be specified for associating P3P policies with content transmitted over other protocols.

P3P allows for policies to be associated with an individual URI or a set of URIs. By associating a separate policy with each URI a site can declare a very precise policy that addresses exactly what data is collected with a particular HTTP request and how that data will be used. However, site management is substantially easier for many sites if they declare a single policy that covers many URIs, or even their entire Web presence.

The P3P specification specifies several methods for referencing a P3P policy reference file, which in turn associates P3P policies with URIs and cookies. XForms can be P3P enabled using any of the methods that are appropriate for the Web site in which they are embedded. Some special considerations regarding forms are addressed in the P3P Specification. [P3P 1.0]

Different P3P policies may be applied to the representation of a form embedded in a containing document to that which is associated with the data submitted via that form. If the form representation is served from a different server than the form is submitted to, it is likely that separate P3P policy reference files and policies will be needed. Typically the form representation causes only *clickstream* data (as defined in [P3P 1.0] section 5.6.4) to be transferred, while a form submission causes much more data to be transferred.

# D 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 **10.1.4 The recalculate Element**.

1 A master dependency directed graph is created as detailed in **D.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 **D.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. The topological sort algorithm is discussed at [Algorithms].

4 The `recalculate` process completes.

## D.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
> **in-degree**: 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 an exception (**4.5.4 The xforms-compute-exception Event**), 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 an exception (**4.5.4 The xforms-compute-exception Event**), 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.

## D.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-Array-Doubling]) can be used to ensure that building the subgraph is performed in time linear in the size of S.

## D.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 an exception (**4.5.4 The xforms-compute-exception Event**) must be thrown, terminating processing.

2 If the vertex corresponds to a computed item, computed expressions are evaluated as follows:

   a `calculate`: If the value of the model item changes, the corresponding instance data is updated and the dirty flag is set.

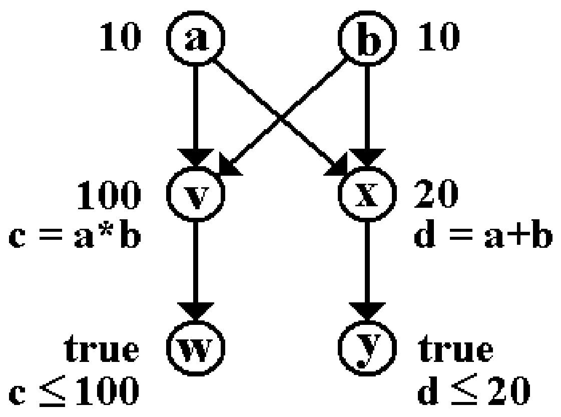
   b `relevant`, `readOnly`, `required`, `constraint`: If any or all of these computed properties change, the new settings are immediately placed into effect for associated form controls.

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.

## D.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 `constraint` attributes and a `nodeset` 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 constraint 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.

## E 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.

## E.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 **E.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.

## E.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.

**Note:**

Additional guidelines for user agent implementation are found at [UAAG 1.0].

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 upperCase" and "user upperCase" 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 upperCase", "cyrillic lowerCase", "cyrillic", "latin", "user upperCase", "user lowerCase"}, then the following `inputmode` values select the following input modes: "cyrillic title" selects "cyrillic", "cyrillic lowerCase" selects "cyrillic lowerCase", "lowerCase cyrillic" selects "cyrillic lowerCase", "latin upperCase" selects "latin", but "upperCase latin" does select "cyrillic upperCase" or "user upperCase" if they correspond to the same input mode, and does not select any input mode if "cyrillic upperCase" and "user upperCase" do not correspond to the same input mode.

## E.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.

### E.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]. Some tokens correspond to the block names in Java class java.lang.Character.UnicodeBlock ([Java Unicode Blocks]) or Unicode Block names. 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 Standard that these script names are taken from is 3.2.

| Input Mode Token | Comments |
|---|---|
| arabic | Unicode script name |
| armenian | Unicode script name |
| bengali | Unicode script name |
| bopomofo | Unicode script name |
| braille | used to input braille patterns (not to indicate a braille input device) |
| buhid | Unicode script name |
| canadianAboriginal | Unicode script name |
| cherokee | Unicode script name |
| cyrillic | Unicode script name |

| Input Mode Token | Comments |
|---|---|
| deseret | Unicode script name |
| devanagari | Unicode script name |
| ethiopic | Unicode script name |
| georgian | Unicode script name |
| greek | Unicode script name |
| gothic | Unicode script name |
| gujarati | Unicode script name |
| gurmukhi | Unicode script name |
| han | Unicode script name |
| hangul | Unicode script name |
| hanja | Subset of 'han' used in writing Korean |
| hanunoo | Unicode script name |
| hebrew | Unicode script name |
| hiragana | Unicode script name (may include other Japanese scripts produced by conversion from hiragana) |
| ipa | International Phonetic Alphabet |
| kanji | Subset of 'han' used in writing Japanese |
| kannada | Unicode script name |
| katakana | Unicode script name (full-width, not half-width) |
| khmer | Unicode script name |
| lao | Unicode script name |
| latin | Unicode script name |
| malayalam | Unicode script name |
| math | mathematical symbols and related characters |
| mongolian | Unicode script name |
| myanmar | Unicode script name |
| ogham | Unicode script name |
| oldItalic | Unicode script name |
| oriya | Unicode script name |
| runic | Unicode script name |
| simplifiedHanzi | Subset of 'han' used in writing Simplified Chinese |
| sinhala | Unicode script name |
| syriac | Unicode script name |
| tagalog | Unicode script name |
| tagbanwa | Unicode script name |
| tamil | Unicode script name |
| telugu | Unicode script name |
| thaana | Unicode script name |
| thai | Unicode script name |
| tibetan | Unicode script name |
| traditionalHanzi | Subset of 'han' used in writing Traditional Chinese |

| Input Mode Token | Comments |
|---|---|
| user | Special value denoting the 'native' input of the user (e.g. to input her name or text in her native language). |
| yi | Unicode script name |

### E.3.2 Modifier Tokens

Modifier tokens can be added to the scripts they apply in order to more closely specify the kind of characters expected in the form control. 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.

| Input Mode Token | Comments |
|---|---|
| lowerCase | lowercase (for bicameral scripts) |
| upperCase | uppercase (for bicameral scripts) |
| titleCase | title case (for bicameral scripts): words start with an upper case letter |
| startUpper | start input with one uppercase letter, then continue with lowercase letters |
| digits | digits of a particular script (e.g. inputmode='thai digits') |
| symbols | symbols, punctuation (suitable for a particular script) |
| predictOn | text prediction switched on (e.g. for running text) |
| predictOff | text prediction switched off (e.g. for passwords) |
| halfWidth | half-width compatibility forms (e.g. Katakana; deprecated) |

## E.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.

## E.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.

| **Caption:** | **`inputmode`** |
|---|---|
| Family name | hiragana |
| (in kana) | katakana |
| Given name | hiragana |
| (in kana) | katakana |
| Zip code | latin digits |
| Address | hiragana |
| (in kana) | katakana |
| Email | latin lowerCase |
| Telephone | latin digits |
| Comments | user predictOn |

# F XForms and Styling (Non-Normative)

This informative section provides a broad outline of new and existing CSS features needed to style XForms content. A future Recommendation from the CSS Working Group will fully develop the specification of these features.

## F.1 Pseudo-classes

A CSS pseudo-class is used to select elements for styling based on information that lies outside of the document tree or that cannot be expressed using the other selectors.

| Name | Defined in: | Relationship to XForms |
|---|---|---|
| `:enabled` & `:disabled` | [CSS3] | Selects any form control bound to a node with the model item property `relevant` evaluating to true or false (respectively). |
| `:required` & `:optional` | TBD | Selects any form control bound to a node with the model item property `required` evaluating to true or false (respectively). |
| `:valid` & `:invalid` | TBD | Selects any form control bound to a node that is currently valid or invalid (respectively), as defined by XForms 1.0. |
| `:read-only` & `:read-write` | TBD | Selects any form control bound to a node with the model item property `readonly` evaluating to true or false (respectively). |

| | | |
|---|---|---|
| `:out-of-range` & `:in-range` | TBD | Selects any form control bound to a node that contains a value the form control is not or is capable of rendering, (respectively). |

This list is not exhaustive; other pseudo-classes may be defined.

## F.2 Pseudo-elements

Pseudo-elements are abstractions about the document tree beyond those specified by the document language. Pseudo-elements do not appear in the DOM; they are used only for purposes of styling.

| Name | Defined in: | Relationship to XForms |
|---|---|---|
| `::value` | TBD | Represents the "active" area of a form control excluding the label; this corresponds in HTML to `input` and other form control elements. This pseudo-element is a child of the form control element, and appears immediately after the required `label` element. |
| `::repeat-item` | TBD | Represents a single item from a repeating sequence. Its position is as a parent to all the elements in a single repeating item. Each `::repeat-item` is associated with a particular instance data node, and is affected by the model item properties (e.g. '`relevant`') found there, as the related style properties will cascade to the child elements. |
| `::repeat-index` | TBD | Represents the current item of a repeating sequence. Its position is as a parent of all the elements in the index repeating item (and as a child to the `::repeat-item` pseudo-element), thus any style declarations applying to this pseudo-element override those on the parent `::repeat-item`. |

This list is not exhaustive; other pseudo-elements may be defined.

## F.3 Examples

The following examples show how basic styling can be accomplished on form controls and repeating structures.

```
@namespace xforms url(http://www.w3.org/2002/xforms);
/* Display a red background on all invalid form controls */
*:invalid { background-color:red; }
/* Display a red asterisk after all required form controls */
*:required::after { content: "*"; color:red; }
/* Do not render non-relevant form controls */
*:disabled { visibility: hidden; }
/* The following declarations cause form controls and their labels
to align neatly, as if a two-column table were used */
xforms|group { display: table; }
xforms|input { display: table-row; }
xforms|input > xforms|label { display: table-cell; }
xforms|input::value { border: thin black solid; display: table-cell; }
/* Display an alert message when appropriate */
*:valid   > xforms|alert { display: none; }
*:invalid > xforms|alert { display: inline; }
/* Display repeat-items with a dashed border */
*::repeat-item { border: dashed; }
/* Display a teal highlight behind the current repeat item */
*::repeat-index { background-color: teal; }
```

## G Complete XForms Examples (Non-Normative)

This section presents complete XForms examples. These and additional examples are maintained at http://www.w3.org/MarkUp/Forms/2002/Examples.

### G.1 XForms in XHTML

```
<!--$Id: index-all.html,v 1.515 2003/10/03 14:35:39 tvraman
Exp $-->
<html xmlns:my="http://commerce.example.com/payment" xm-
lns:ev="http://www.w3.org/2001/xml-events" xmlns:xsd="ht-
tp://www.w3.org/2001/XMLSchema" xmlns:xforms="ht-
tp://www.w3.org/2002/xforms" xmlns="ht-
tp://www.w3.org/2002/06/xhtml2">
<head>
<title xml:lang="fr">XForms en XHTML</title>
<xforms:model schema="payschema.xsd">
<xforms:instance>
<my:payment as="credit">
<my:cc />
<my:exp />
</my:payment>
</xforms:instance>
<xforms:submission action="http://www.example.com/buy.rb"
method="post" id="s00" />
```

```
<xforms:bind nodeset="my:cc" relevant="../@as='credit'" re-
quired="true()" />
<xforms:bind nodeset="my:exp" relevant="../@as='credit'"
required="true()" />
</xforms:model>
</head>
<body>
    ...
<group xmlns="http://www.w3.org/2002/xforms">
<trigger>
<label>Français</label>
<toggle case="fr" ev:event="DOMActivate" />
</trigger>
<trigger>
<label>English</label>
<toggle case="en" ev:event="DOMActivate" />
</trigger>
</group>
<switch xmlns="http://www.w3.org/2002/xforms">
<case id="fr">
<select1 ref="@as">
<label xml:lang="fr">Choisissez un mode de paiement</label>
<choices>
<item>
<label xml:lang="fr">Comptant</label>
<value>cash</value>
<message level="modeless" ev:event="xforms-select"
xml:lang="fr">
Ne pas envoyer d'argent comptant par la poste.</message>
</item>
<item>
<label xml:lang="fr">Carte bancaire</label>
<value>credit</value>
</item>
</choices>
</select1>
<input ref="my:cc">
<label xml:lang="fr">Numéro de carte bancaire</label>
<alert xml:lang="fr">Saississez un numéro de carte ban-
caire en cours
           (séparez par un espace ou un trait d'uni-
on chaque groupe de chiffres)</alert>
</input>
<input ref="my:exp">
```

```
<label xml:lang="fr">Date d'échéance</label>
</input>
<submit submission="s00">
<label xml:lang="fr">Achetez</label>
</submit>
</case>
<case id="en">
<select1 ref="@as">
<label xml:lang="en">Select Payment Method</label>
<choices>
<item>
<label xml:lang="en">Cash</label>
<value>cash</value>
<message level="modeless" ev:event="xforms-select"
xml:lang="en">
Please do not mail cash.</message>
</item>
<item>
<label xml:lang="en">Credit</label>
<value>credit</value>
</item>
</choices>
</select1>
<input ref="my:cc">
<label xml:lang="en">Credit Card Number</label>
<alert xml:lang="en">Please specify a valid credit card num-
ber
            (use spaces or hyphens between di-
git groups)</alert>
</input>
<input ref="my:exp">
<label xml:lang="en">Expiration Date</label>
</input>
<submit submission="s00">
<label xml:lang="en">Buy</label>
</submit>
</case>
</switch>
    ...
</body>
</html>
```

Schema file `payschema.xsd`:

```
<!-- payschema.xsd -->
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:my="http://commerce.example.com/payment" target-
Namespace="http://commerce.example.com/payment" elementForm-
Default="qualified">
<xsd:element name="payment">
<xsd:complexType name="payment">
<xsd:sequence minOccurs="0" maxOccurs="unbounded">
<xsd:choice>
<xsd:element ref="my:cc" />
<xsd:element ref="my:exp" />
</xsd:choice>
</xsd:sequence>
<xsd:attribute name="as" type="my:paymentAs" />
</xsd:complexType>
</xsd:element>
<xsd:element name="cc" type="my:cc" />
<xsd:element name="exp" type="xsd:gYearMonth" />
<xsd:simpleType name="cc">
<xsd:restriction base="xsd:string">
<xsd:pattern value="\s*((\d+)[-\s])+([\d]+)\s*" />
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="paymentAs">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="cash" />
<xsd:enumeration value="credit" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

## G.2 Editing Hierarchical Bookmarks Using XForms

```
<html xmlns:ev="http://www.w3.org/2001/xml-events" xm-
lns:my="http://commerce.example.com/payment" xmlns:xsd="ht-
tp://www.w3.org/2001/XMLSchema" xmlns:xforms="ht-
tp://www.w3.org/2002/xforms" xmlns="ht-
tp://www.w3.org/2002/06/xhtml2" xml:lang="en">
<head>
<style type="text/css">
xforms|input.editField {
font-weight:bold; font-size:20px; width:500px
      }
xforms|label.sectionLabel {
font-weight:bold; color:white; background-color:blue
```

```
      }
xforms|submit {
font-family: Arial; font-size: 20px; font-style: bold; col-
or: red
      }
</style>
<title>Editing Hierarchical Bookmarks In X-Smiles </title>
<xforms:model id="bookmarks">
<xforms:instance src="bookmarks.xml" />
<xforms:submission id="s01" method="post" action="http://ex-
amples.com/" />
</xforms:model>
</head>
<body>
<xforms:repeat nodeset="section" id="repeatSections">
<xforms:input ref="@name" class="editField">
<xforms:label class="sectionLabel">Section</xforms:label>
</xforms:input>
<!-- BOOKMARK REPEAT START -->
<xforms:repeat nodeset="bookmark" id="repeatBookmarks">
<xforms:input ref="@name">
<xforms:label>Bookmark name</xforms:label>
</xforms:input>
<xforms:input ref="@href">
<xforms:label>URL</xforms:label>
</xforms:input>
</xforms:repeat>
</xforms:repeat>
<p>
<!-- INSERT BOOKMARK BUTTON -->
<xforms:trigger id="insertbutton">
<xforms:label>Insert bookmark</xforms:label>
<xforms:insert nodeset="section[index('repeatSections')]/book-
mark" at="index('repeatBookmarks')" position="after"
ev:event="DOMActivate" />
</xforms:trigger>
<!-- DELETE BOOKMARK BUTTON -->
<xforms:trigger id="delete">
<xforms:label>Delete bookmark</xforms:label>
<xforms:delete nodeset="section[index('repeatSections')]/book-
mark" at="index('repeatBookmarks')" ev:event="DOMActivate"
/>
</xforms:trigger>
</p>
```

```
<p>
<!-- INSERT SECTION BUTTON -->
<xforms:trigger id="insertsectionbutton">
<xforms:label>Insert section</xforms:label>
<xforms:insert nodeset="section" at="index('repeatSections')"
position="after" ev:event="DOMActivate" />
</xforms:trigger>
<!-- DELETE SECTION BUTTON -->
<xforms:trigger id="deletesectionbutton">
<xforms:label>Delete section</xforms:label>
<xforms:delete nodeset="section" at="index('repeatSections')"
ev:event="DOMActivate" />
</xforms:trigger>
</p>
<!-- SUBMIT BUTTON -->
<xforms:submit submission="s01">
<xforms:label>Save</xforms:label>
<xforms:hint>Click to submit</xforms:hint>
</xforms:submit>
</body>
</html>
```

Initial instance file `bookmarks.xml`:

```
<!--This is the bookmarks.xml file -->
<bookmarks>
<section name="main">
<bookmark href="http://www.example.com/xforms.xml" name="Main
page" />
</section>
<section name="demos">
<bookmark href="http://www.example.com/demo/images.fo"
name="images" />
<bookmark href="http://www.example.com/demo/xf-ecma.xml"
name="ecma" />
<bookmark href="http://www.example.com/demo/sip.fo"
name="sip" />
</section>
<section name="XForms">
<bookmark href="file:///C/source/xmlevents.xml" name="XML
events" />
<bookmark href="file:///C/source/model3.xml" name="model3"
/>
<bookmark href="file:///C/source/repeat.fo" name="repeat"
/>
```

```
</section>
</bookmarks>
```

## G.3 Survey Using XForms and SVG

The following example shows one possible way of integrating XForms with [SVG 1.1]. Note that the complete set of rules for integrating XForms and SVG are not fully specified at the time this specification was published. Future versions of the XForms, SVG, or other W3C specifications might define more complete rules for integrating XForms and SVG which might not be compatible with the example below.

Note that the example below does not use SVG's `switch` and `requiredExtensions` features, which are commonly used in conjunction with `foreignObject`.

```
<!-- <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"> -->
<svg xmlns:s="http://example.com/survey" xmlns:ev="ht-
tp://www.w3.org/2001/xml-events" xmlns:xforms="ht-
tp://www.w3.org/2002/xforms" xmlns:xlink="ht-
tp://www.w3.org/1999/xlink" xmlns="ht-
tp://www.w3.org/2000/svg" width="700px" height="600px"
viewBox="0 0 700 600">
<defs>
<polygon id="bullet" points="-30,-30, -10,-10, -20,10"
fill="#007138" />
<xforms:model id="form1" schema="surveyschema.xsd">
<xforms:instance id="instance1">
<s:survey xmlns="http://example.com/survey">
<s:drink>none</s:drink>
<s:espressoPrefs>
<s:numberPerWeek>0</s:numberPerWeek>
<s:sugar>0</s:sugar>
<s:lemon>Always</s:lemon>
</s:espressoPrefs>
</s:survey>
</xforms:instance>
<xforms:submission id="submit1" method="post" action="ht-
tp://www.example.org/surveyhandler" />
</xforms:model>
</defs>
<title>Espresso survey</title>
<desc>Sample SVG and XForms - espresso customer survey</desc>
<g>
```

```
<text x="50" y="70" font-size="40" font-family="Arial Black,
sans-serif" font-weight="900">Customer Survey: Es-
presso</text>
<g font-family="Arial, Helvetica, sans-serif" font-size="18">
<foreignObject x="80" y="150" width="250" height="40">
<xforms:select1 appearance="minimal" model="form1"
ref="s:drink">
<xforms:label>
<g transform="translate(80, 140)">
<use xlink:href="#bullet" />
<text>Your usual coffee drink is:</text>
</g>
</xforms:label>
<xforms:item>
<xforms:label>Rich, dark espresso</xforms:label>
<xforms:value>espresso</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Creamy cappuccino</xforms:label>
<xforms:value>cappuccino</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Long, milky lattï¿½</xforms:label>
<xforms:value>lattï¿½</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Don't like coffee!</xforms:label>
<xforms:value>none</xforms:value>
</xforms:item>
</xforms:select1>
</foreignObject>
<foreignObject x="80" y="240" width="250" height="40">
<xforms:range model="form1" start="0" end="30" step="5"
ref="s:espressoPrefs/s:numberPerWeek">
<xforms:label>
<g transform="translate(80, 230)">
<use xlink:href="#bullet" />
<text>Shots of espresso per week:</text>
</g>
</xforms:label>
</xforms:range>
</foreignObject>
<foreignObject x="80" y="350" width="250" height="40">
<xforms:select model="form1" ref="s:espressoPrefs/s:sugar">
```

```
<xforms:label>
<g transform="translate(80, 340)">
<use xlink:href="#bullet" />
<text>Sugar?</text>
</g>
</xforms:label>
<xforms:item>
<xforms:label>Yes</xforms:label>
<xforms:value>X</xforms:value>
</xforms:item>
</xforms:select>
</foreignObject>
<foreignObject x="80" y="420" width="250" height="90">
<xforms:select1 appearance="full" model="form1" ref="s:es-
pressoPrefs/s:lemon">
<xforms:label>
<g transform="translate(80, 410)">
<use xlink:href="#bullet" />
<text>Lemon?</text>
</g>
</xforms:label>
<xforms:item>
<xforms:label>Required for the full experience</xforms:label>
<xforms:value>Always</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Whatever</xforms:label>
<xforms:value>Indifferent</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Keep that citrus to yourself</xforms:label>
<xforms:value>Never</xforms:value>
</xforms:item>
</xforms:select1>
</foreignObject>
</g>
<use xlink:href="#bullet" x="101" y="64" trans-
form="scale(7,3)" />
<foreignObject y="150" x="500" height="60" width="100">
<xforms:submit model="form1">
<xforms:label>Send survey</xforms:label>
</xforms:submit>
</foreignObject>
<!--- keep the graphics data out of this example listing -->
```

```
<image xlink:href="espresso.svg" x="400" y="230" width="280"
height="270" />
</g>
</svg>
```

## H Changelog (Non-Normative)

This section summarizes changes since the 1 August XForms 1.0 Proposed Recommendation.

- Minor editorial adjustments.
- Added disclaimer text to SVG example.

## I Acknowledgments (Non-Normative)

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

- Steven Pemberton, W3C/CWI (*Co-chair*)
- Sebastian Schnitzenbaumer, SAP/Mozquito (*Co-chair*)
- Rob McDougall, Adobe
- Micah Dubinko, Cardiff (*Editor*)
- Mikko Honkala, Helsinki University Of Technology
- Roland Merrick, IBM (*Editor*)
- T. V. Raman, IBM (*Editor*)
- David Landwehr, Novell
- Kenneth Sklander, Novell
- John Boyer, PureEdge Solutions Inc.
- Thierry Michel, W3C (*W3C Team Contact*)
- Leigh Klotz, Xerox (*Editor*)
- Mark Birbeck, x-port.net Ltd. (*Invited Expert*)
- Subramanian Peruvemba, Oracle Corp.
- Mark Seaborne, Origo Services Limited
- Daniel Vogelheim, Sun Microsystems

Former Working Group participants:

- Peter Stark, Ericsson
- Vincent Godefroy, EDF R&D
- Davanum Srinivas, Computer Associates
- Doug Dominiak, Openwave
- Frank Boumphrey, HTML Writer's Guild
- Linda Bucsay Welsh, Intel
- Gavin McKenzie, JetForm Corporation
- John McCarthy, Lawrence Berkeley National Laboratory
- Frank Olken, Lawrence Berkeley National Laboratory
- Ray Waldin, Lexica, LLC
- Tantek Çelik, Microsoft
- Josef Dietl, Mozquito Technologies
- Dave Hyatt, Netscape/AOL
- Eric Pollmann, Netscape/AOL
- Kazunari Kubota, NTT DoCoMo, Inc.
- Kaori Nakai, NTT DoCoMo, Inc.
- Tom Butcher, OpenDesign
- Ted Wugofski, Openwave
- Jeremy Chone, Oracle
- K. P. Lee, Philips
- Panagiotis Reveliotis, Philips
- Roli Wendorf, Philips
- David Cleary, Progress Software
- Mike Mansell, PureEdge Solutions Inc
- Michael Fergusson, SoftQuad
- Zoe Lacroix, SurroMed, Inc.
- Dave Navarro, WebGeek Inc.

- Masayasu Ishikawa, W3C (*Team Contact until September 2001*)
- Dave Raggett, W3C/Openwave (*Team Contact until December 2000*)
- Larry Masinter, Xerox

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), Linda Bucsay Welsh (until April 2001), and Josef Dietl (until October 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 parts of the XForms events and action handler processing models as well as sections on the recalculation sequence algorithm. Finally, we would like to thank members of the public www-forms@w3.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.

## J 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 XML Schema portion of the Appendix was rendered into HTML with the xmlverbatim XSLT stylesheet (used with permission). 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.