XForms 1.0

W3C Working Draft 08 June 2001

This version:
http://www.w3.org/TR/2001/WD-xforms-20010608 (One big file, Zip archive)
Latest version:
http://www.w3.org/TR/xforms/
Previous versions:
http://www.w3.org/TR/2001/WD-xforms-20010216
Editors:
Micah Dubinko, Cardiff <mdubinko@Cardiff.com>
Josef Dietl, Mozquito Technologies <josef@mozquito.com>
Roland Merrick, IBM <Roland_Merrick@uk.ibm.com>
Dave Raggett, W3C/Openwave <dsr@w3.org>
T. V. Raman, IBM <tvraman@almaden.ibm.com>
Linda Bucsay Welsh, Intel <linda@intel.com>

Abstract

Forms were introduced into HTML in 1993. Since then they have become a critical part of the Web. The existing mechanisms in HTML for forms are now outdated, and W3C has started work on developing an effective replacement. This document outlines the requirements for "XForms", W3C's name for the next generation of Web forms.

Status of this Document

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

This is a Working Draft that incorporates new material agreed upon at the Boston face to face meeting, including the adoption of XML Schema replacing XForms Simple Syntax, as well as initial efforts at modularizing XForms and additional feedback from outside sources. Interested parties are encouraged to provide additional feedback and comments.

This document is a W3C Working Draft for review by W3C members and other

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

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

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

Table of Contents

Appendices

---

1 About the XForms 1.0 Specification

1.1 Background

Forms are an important part of the Web, and they continue to be the primary means of interactivity used by many Web sites. Web applications and eCommerce solutions have sparked the demand for better Web forms with richer interactions. XForms are the response to this demand--extended analysis, followed by the creation of a new platform-independent markup language for online interaction between an XForms Processor and a remote entity. XForms are the successor to XHTML forms, and benefit from the lessons learned in the years of HTML forms implementation experience.

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 and becomes more and more technical and specific towards the end. For quick access to information, a general table of contents, specific tables of contents at the beginning of each section, and an index provide easy navigation, in both the electronic and printed versions.

The specification has been written with two modes of presentation in mind: electronic and printed. In case of a discrepancy, the electronic version is

considered the authoritative version of the document.

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 includes a brief tutorial on XForms and a discussion of design principles behind XForms.

Chapters 3 and up

> XForms reference manual. The bulk of the reference manual consists of the specification of XForms. This reference defines what may go into 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 optional function libraries, references, a change history, and other useful information.

1.4 Documentation Conventions

The following highlighting and typography is used to present technical material in this document and other documents from the XForms Working Group:

Special terms are defined in their own chapter; hyperlinks connect uses of the term to the definition.

Throughout this document, the namespace prefixes "`xform:`", "`xsd:`", and "`xsi:`" are used to denote the XForms, XML Schema, and XML Schema for Instances namespaces respectively. This is by convention only; any namespace prefix may be used in practice.

Official terms are defined in the following manner: [Definition: You can find most **terms** in the chapter **3 Terminology**]. Links to terms may be specially highlighted in the text.

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

Example: XML Syntax Representation `<example>`

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

count = integer - description of this attribute
size = (small | medium | large) : medium - description of this attribute

Non-normative short examples are set off typographically:

Example Item

or

```
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 highlighting is used for non-normative commentary:

Note:

A general admonition to readers.

Editorial note: Editorial Note Name	
Editorial commentary.	

Issue (issue-id):

Issue-Name

A specific issue to which input from readers is requested.

2 Concepts

This informative chapter provides an easily approachable description of the design of XForms, describing the major components and how they relate. Not every feature of XForms is covered here. For a complete, normative description of XForms, refer to the remainder of this document.

2.1 Purpose and Presentation

For explanatory purposes, a form can be considered to consist of 'purpose', 'presentation', and 'data'. Some examples:

Purpose	Presentation	Data
Data collection	Arrangement of form controls	Registration information
Time card	How dates are entered	Days and hours worked
Order form	How to render the form controls on small devices	Order, shipping, and payment info
Information Please	How the form integrates with a Web site	User contact information

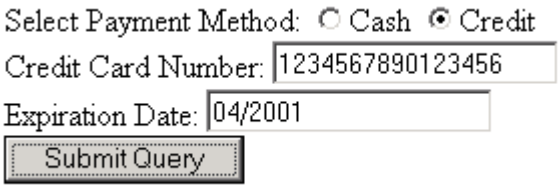
The design of existing Web forms didn't separate the purpose from the presentation of a form, and additionally offered only a restricted representation for data captured through the form. This is the primary difference between XForms and previous form technologies.

2.2 Current Approach: XHTML

Take for instance a simple eCommerce form authored in XHTML 1.0:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
          "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<body>
<form action="http://example.com/submit" method="post">
  <span>Select Payment Method: </span>
  <input type="radio" name="paytype" value="cash">Cash</input>
  <input type="radio" name="paytype" value="credit">Credit</input><br/>
  <label>Credit Card Number: <input type="text" name="cc"/></label><br/
  <label>Expiration Date: <input type="text" name="exp"/></label><br/>
  <input type="submit"/>
</form>
</body>
</html>
```

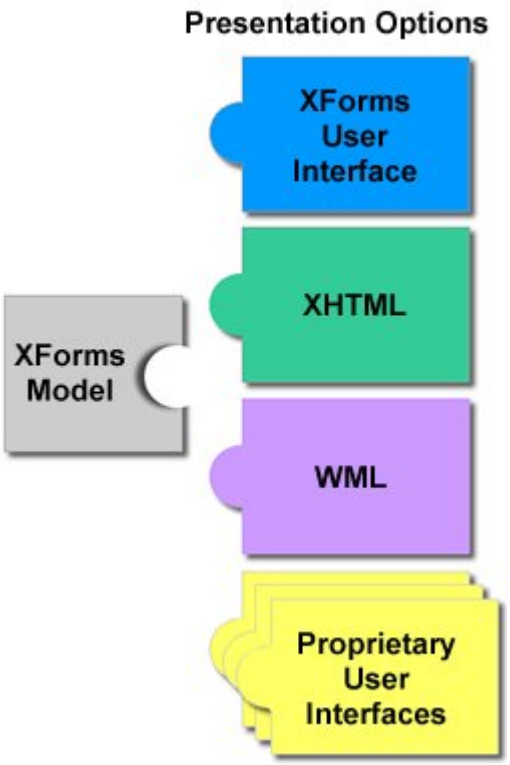
A browser might render this form as follows:



This form makes no effort to separate purpose (data collection semantics) from presentation (the `<input>` form controls), and offers no control over the basic name/value formatting of the resulting data. XForms greatly improves the expressive capabilities of electronic forms.

2.3 Stepping Up to XForms

XForms are comprised of separate sections that describe what the form does, and how the form is to be presented. This allows for flexible presentation options, making it possible for classic XHTML form controls, as well as other form control sets such as WML to be leveraged, as shown here.



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

```
<xform:xform>
  <xform:submitInfo target="http://example.com/submit" method="..."/>
</xform:xform>
```

With these changes to the containing document, the previous example could be rewritten like this:

```
<selectOne xmlns="http://www.w3.org/2001/06/xforms" ref="paytype">
  <caption>Select Payment Method</caption>
  <choices>
    <item value="cash">Cash</item>
    <item value="credit">Credit</item>
  </choices>
</selectOne>
<textbox xmlns="http://www.w3.org/2001/06/xforms" ref="cc">
  <caption>Credit Card Number</caption>
</textbox>
<textbox xmlns="http://www.w3.org/2001/06/xforms" ref="exp">
```

```
    <caption>Expiration Date</caption>
  </textbox>
  <submit xmlns="http://www.w3.org/2001/06/xforms"/>
```

Notice the following features of this design:

- The user interface is not hard-coded to use radio buttons. Different devices (such as a voice browser) can render the concept of "selectOne" as appropriate.
- Form controls always have captions directly associated with them, as child elements.
- There is no need for an enclosing `form` element.
- Element names for form controls have been changed: `textbox` is a specific element, rather than a `type` attribute on `input`, as in XHTML.
- Data entered through the form controls ends up submitted as XML.

With these changes, the XForms Processor will be able to directly submit XML instance data. The XML is constructed by creating a root element with child elements reflecting the names given to each form control. For instance, the submitted data might look like this:

```
<!-- envelope, generated separately -->
<Envelope>
  <Body>
```

```
    <!-- serialized instance data -->
    <paytype>Credit</paytype>
    <cc>12354677890123456</cc>
    <exp>04-2001</exp>
```

```
    <!-- envelope, generated separately -->
    </Body>
</Envelope>
```

2.4 Providing XML Instance Data

Understandably, authors will often desire greater control over exact construction of the submitted instance data. One common case might be submitting to a server XML data that is validated against a predefined DTD or XML Schema.

XForms keeps track of the state of the partially filled form through instance data, which provides an outline of the desired XML data, including namespace information. The instance data starts off with the initial values for the form, is updated as the user fills the form, and eventually is serialized and submitted. The initial instance data is taken from the `instance` element inside the `xform` element, defined as follows:

```
<xform:xform>
  <xform:submitInfo target="http://example.com/submit" method="..."/>
  <xform:instance>
    <payment type="credit" xmlns="http://commerce.example.com/payment">
      <cc/>
      <exp/>
    </payment>
  </xform:instance>
</xform:xform>
```

This design has features worth calling out:

- There is complete flexibility in the structure of the XML. Notice that the item `paytype` is now expressed as an attribute `type` of the element `payment`.
- Empty elements `cc` and `exp` serve as placeholders in the XML structure, and will be filled in with form data provided by the user.
- An initial value (`"credit"`) for the form control is provided through the `type` attribute in the instance data. In the submitted XML, this initial value will be replaced by the user input, if any.
- The instance data provides a unique namespace, which will be used when the XML gets submitted.

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

```
  <selectOne ref="payment/@type">
  ...
  <inputText ref="payment/cc">
  ...
  <inputText ref="payment/exp">
```

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

2.5 The XForms Model

The earlier XHTML form in **2.2 Current Approach: XHTML**. Even in this short form, there are several aspects that would be desirable to express, but would only be possible through the addition of unstructured script code:

- The credit card information fields `cc` and `exp` are only relevant if the "Credit" option is chosen in the `paytype` field.
- The credit card information fields `cc` and `exp` should be required when the "Credit" option is chosen in the `paytype` field.
- The field `cc` should accept digits only, and should have exactly 14, 15, or 16 digits.
- The field `exp` should accept only valid month/date combinations.

By specifying a 3rd component, the XForms Model, authors can include rich declarative datatype and validation information in forms.

Editorial note: MJD	
The examples here are sketchy out of necessity; this section will need to be rewritten after the Schema Basic task force delivers its syntax recommendations.	

An XForms Model consists of model items, which include XML Schema datatype information [XML Schema part 2] as well as properties specific to XForms.

```
<!-- add to the cc model item the following: -->
relevant="value('payment/@type') == 'credit'"
required="true"
datatype of "xform:string"
facet pattern of "\d{14,16}"

<!-- add to the exp model item the following: -->
relevant="value('payment/@type') == 'credit'"
required="true"
datatype of "xform:gYearMonth"
```

2.6 Multiple Forms per Document

XForms places no limits on the number of individual forms that can be placed in a single containing document. When multiple forms share the same containing document, multiple `xform` elements are needed. The first `xform` element may skip a unique `id` attribute (as have all the examples above), but subsequent `xform` elements require an `id` so that they can be referenced from elsewhere in the containing document.

The other side of the equation is that form controls throughout the document need to specify which `xform` element is associated with the instance data to which they bind. This is accomplished through an `xform` attribute alongside the `ref` attribute. The default for the `xform` attribute is to refer to the first `xform` element in document order.

To add a second form, an opinion poll, to our commerce example, the following would be authored in the head section of the XHTML:

```
<xform:xform>
  <xform:submitInfo target="http://example.com/submit" method="..."/>
  <xform:instance>
     ...payment instance data...
  </xform:instance>
</xform:xform>

<xform:xform id="poll">
  <xform:submitInfo target="http://example.com/poll" method="..."/>
</xform:xform>
```

Additionally, the following form control markup in the body:

```
<selectOne ref="pollOption" xform="poll"  xmlns="http://www.w3.org/2001
  <caption>How useful is this page to you?</caption>
  <choices>
    <item value="0">Not at all helpful</item>
    <item value="1">Barely helpful</item>
    <item value="2">Somewhat helpful</item>
    <item value="3">Very helpful</item>
  </choices>
</selectOne>
<submit xform="poll"  xmlns="http://www.w3.org/2001/06/xforms"/>
```

The main difference to note here is the use of `xform="poll"`, which identifies which form the form control binds to.

2.7 Additional User Interface Capabilities

The visual layout appearance of the initial XHTML forms such as the above example (**2.2 Current Approach: XHTML**) leaves much to be desired.

```
Need extended UI example here
```

2.8 Complete Document

This chapter presented various bits and pieces of XForms as a tool to help readers understand the design. Presented here is the entire XHTML+XForms document presented in one segment.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
          "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xform="http://www.w3.org/2001/06/xforms"
      xml:lang="en">
<head>
  <title>XForms in XHTML</title>

  <xform:xform>
    <xform:submitInfo target="http://example.com/submit" method="..."/>
    <xform:instance>
      <payment type="credit" xmlns="http://commerce.wizard">
        <cc/>
        <exp/>
      </payment>
    </xform:instance>
  </xform:xform>

  <xform:xform id="poll">
    <xform:submitInfo target="http://example.com/poll" method="..."/>
  </xform:xform>

</head>
<body>
```

```
        ... include advanced UI markup here ...
    </body>
    </html>
```

3 Terminology

binding

[Definition: The connection between a form control and a model item and an instance data item, represented as a binding expression.]

binding expression

[Definition: An XPath addressing expression used by the binding to connect form controls to other parts of XForms.]

computed expression

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

containing document

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

datatype

[Definition: 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. This definition is taken from XML Schema [XML Schema part 2].]

facet

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

form control

[Definition: A user interface control or "widget" that serves as a point of user interaction.]

instance data

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

instance data item

[Definition: An internal representation of the value and state of a single piece of data corresponding to a Schema simpleType, constrained by the definition of a model item.]

lexical space

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

model item

[Definition: An abstract unit of data-collection within the XForms Model, which defines a XML Schema datatype and possibly other form-specific constraints on a single piece of collected data.]

model item property

[Definition: A single, XForms-specific defining aspect of a model item..]

value space

[Definition: 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. This definition is taken from XML Schema [XML Schema part 2].]

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 the XForms specification.]

4 Datatypes

4.1 XML Schema Datatypes

XForms includes all XML Schema datatypes, including the concepts of value space and lexical space, and all constraining facets, as specified in [XML Schema part 2]. For reference, these are

Built-in primitive types:

duration
dateTime

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

Built-in derived types:

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

The Schema for XForms derives new types for all the above datatypes, placing no additional restrictions on the allowed value space, but including them in the XForms namespace.

Issue (datatype-identifiers):

One requirement is for XForms to include unique identifiers for each datatype listed here. We believe the facilities in XML Schema are sufficient for this, but welcome feedback on this issue.

Issue (ieee-datatypes):

There is concern that IEEE floating point, as used by the datatypes `xsd:float` and `xsd:double` may be difficult or impossible to implement on small devices. XPath (see **6 XPath Expressions in XForms**) also uses IEEE representation for numbers, and must be taken into consideration as to resolution of this conflict.

Issue (xml-datatypes):

Several datatypes, namely `xsd:QName`, `xsd:NOTATION`, `xsd:Name`, `xsd:NCName`, `xsd:ID`, `xsd:IDREF`, `xsd:IDREFS`, `xsd:ENTITY`, and `xsd:ENTITES` are highly specific to XML, and perhaps not as useful in XForms. Particularly on small devices, the incremental cost of supporting these datatypes might be excessive.

Issue (pattern-facet):

Fully implementing the `pattern` facet might be too expensive on small devices.

Issue (mask-facet):

Previous Working Drafts of XForms specified a `mask` facet, with less implementation cost than the Schema `pattern` facet. What are our options for adding an entirely new facet not defined in XML Schema?

The mask facet was defined as follows:

Example: Mask Facet

XML Schema has defined a Regular Expression language which is "similar to the regular expression language used in the Perl Programming language", and can be applied to most built-in datatypes. Regular expression syntax, however, is considered complex by some. Therefore, XForms defines the concept of a `mask` facet. All `mask` facets are convertible into `pattern`s.

XML schema allows multiple `pattern` facets to be specified. Similarly, multiple `mask` or `pattern` facets, but not a mixture, are permitted.

XForms `mask` uses the syntax and processing from [WML1.3] `format`. Some examples:

- `A` matches "A", "X", "$", "%", or "."
- `a` matches "a", "x", "$", "%", or "."
- `X` matches "A", "X", "$", "%", ".", or "4"
- `x` matches "a", "x", "$", "%", ".", or "4"
- `N` matches "0", "4", or "7"
- `3N` matches "0", "63", or "999" but not "1234" (Note: only allowed at end of mask)

- `*X` matches "$", "3.0", or "ABCDEFG" (Note: only allowed at end of mask)
- `\` causes the next literal character to be inserted into the mask
- `NNN\-NNNN` matches "123-4567" but not "1234567"

As with WML `format` processing, an XForms Processor must ignore invalid `mask`s.

Issue (now-facet):

Previous Working Drafts of XForms specified "dynamic facets" that could be reevaluated at arbitrary times. One benefit of that approach was that a `now()` expression could be used as a constraining facet on date/time datatypes. What are our options for including similar functionality within the framework of XML Schema datatypes?

4.2 XForms Datatypes

The Schema for XForms derives the following types for use within forms:

4.2.1 currency

The XForms datatype `currency` is derived from the XForms datatype `xform:string`. A `pattern` facet with a value of `[A-Z]{3}` is specified.

Valid currency values are specified in [ISO 4217].

Note:

Since new currencies may appear at any time, the currency values as defined in [ISO 4217] are not strictly enforced.

- **Examples:**A value of 'Japanese Yen' would be represented as:

  JPY

- A value 'US Dollars' would be represented as:

  USD

4.2.2 monetary

Forms often deal with monetary values. In many cases the currency type in use is well-known, and not even needed in the instance data. When an explicit currency identifier is needed, authors can freely use separate `xform:decimal` and `xform:currency` values to accomplish this. For a unified approach, a monetary datatype would combine, in a single lexical space, both a currency code and a decimal value.

Issue (monetary-datatype):

Is such a datatype needed in XForms? If yes, how can it be defined in terms of XML Schema?

- **Examples:** A value of 42 Indonesian Rupiahs would be represented as:

  42IDR

- A value of 4.37 Euro would be represented as:

  4.37EUR

5 The XForms Model

5.1 Introduction

Chapter **4 Datatypes** described how XForms uses the XML Schema datatyping system, which can constrain the value space of datatypes that can be used in data collection. This chapter introduces a different set of properties, called model item properties, which define XForms-specific behaviors and metadata useful for data collection.

5.2 Model Item Properties

Model item properties fall into two basic categories:

- Computed expressions are XPath expressions that provide a value to the XForms Processor. The value is recomputed at certain times, according to the XForms Processing Model (see **11 Processing Model**).
- All other properties are fixed, static values that the XForms Processor evaluates only once.

The following properties are available for all model items, and their syntax is explained throughout this chapter. For each property the following information is provided:

Description
Computed Expression (yes or no)
Legal Values
Default Value
Additional descriptive text

5.2.1 name

Description: provides a specific name for the model item.

Computed Expression: No

Legal Values: only values of type `xsd:NCName`

Default Value: none.

Authors can associate a human-readable name with a model item through the use of the `name` property. Each `name` should be unique within the scope of the XForms Model where it is declared.

5.2.2 type

Description: assigns a Schema datatype.

Computed Expression: No

Legal Values: any `xsd:QName` representing a Schema datatype.

Default Value: `xsd:anyType`

The concept of typed data is important to forms. The assignment of a particular datatype to a model item affects validation of the data it can accept, as well as affecting which form controls to which it can bind.

> **Note:**
>
> The XForms Model uses properties "name" and "type" as in XML Schema; the concrete syntax used to define XForm Models, and consequently the use of these properties will be made concrete in a forthcoming revision of this Working Draft.

5.2.3 readOnly

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

Computed Expression: Yes

Legal Values: any expression is convertible to `boolean`.

Default Value: `false`.

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

In addition to restricting value changes, the `readOnly` property provides a hint to the XForms User Interface. Form controls bound to a model item with the `readOnly` property should indicate that entering or changing the value is not allowed. The hint provided has no effect on visibility, focus, or navigation order.

5.2.4 required

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

Computed Expression: Yes

Legal Values: any expression that is convertible to `boolean`

Default Value: `false`.

Often forms require certain values to be entered. This may be a static requirement, or may only be the case if some condition is satisfied. When evaluating to `true`, this property indicates that a non-empty instance data item is required before a submission of instance data can occur. Non-empty is defined as:

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

Except as noted below, the `required` 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 a unique indication that a form control is required, and may provide immediate feedback, including limiting navigation, for required form controls.

> **Issue (issue-default-default):**
>
> It might be useful to set the default for the required attribute for an entire XForms Model. What should the default default be? How could we assign a default for a single XForms Model? This could apply to other attributes as well, e.g. readOnly, etc...

The chapter **11 Processing Model** contains details on how the XForms Processor enforces required values.

5.2.5 relevant

Description: indicates whether the model item is currently relevant to the rest of the XForms Model. XForms Processors would typically not render an associated form control, including children, when the value is `false`.

Computed Expression: Yes

Legal Values: any expression is convertible to `boolean`

Default Value: `true`.

Many forms have fields dependent 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.

When evaluating to `true`, this property indicates that the XForms Processor should render a form control, and conversely, when evaluating to `false`, indicates that the form control should not be rendered.

The `relevant` 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.

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

	`required="true"`	`required="false"`
`relevant="true"`	The form control (and any children) should be visible or available to the user. The XForms User Interface may indicate that a value is required.	The form control (and any children) should 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) should be hidden or unavailable 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.	The form control (and any children) should be hidden or unavailable to the user. Entering a value or obtaining focus should not be allowed.

5.2.6 calculate

Description: indicates that the instance data item associated with the model item is to be dynamically calculated.

Computed Expression: Yes

Legal Values: any expression is convertible to an XPath datatype compatible with the associated XML Schema datatype.

Default Value: none.

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

5.2.7 priority

Description: indicates the relative priority for calculations of the model item.

Computed Expression: No

Legal Values: any expression that is convertible to an integer in the range 0-32767.

Default Value: 0.

For model items that are calculated, this optional property specifies a calculation order. The XForms Processing Model uses this property to determine the calculation order for multiple calculations.

5.2.8 validate

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

Computed Expression: Yes

Legal Values: any expression that is convertible to `boolean`

Default Value: `true`.

An XForms Model may include model items that need to be validated. When evaluating to `true`, indicates that the model item is considered valid. The chapter **11 Processing Model** describes details such as immediate validation vs. onsubmit validation.

Computed expressions used here are not restricted to examining the instance data item they are invoked on. XPath, plus the extensions in this chapter, provide the means to traverse the instance data, as well as call-outs to external script, enabling potentially complex validations.

The XForms User Interface may indicate whether a form control is currently valid or invalid.

> **Issue (issue-cascade):**
>
> Will the `validate` property be evaluated on all the parent or child model items whenever a value changes? We need to make sure that inter-model item constraints will get evaluated.

5.3 Using Datatypes in the XForms Model

The following section is being rewritten with the guidance of the XML Schema Working Group. In its current state, it is an informative listing of the functionality that we are planning in XForms 1.0, with illustrative examples of similar functionality in XML Schema. A subsequent Working Draft will contain normative details on how the functionality is described in terms of XForms.

5.3.1 Atomic Datatype

At the simplest level, it is necessary to associate a datatype with a model item. This has the effect of restricting the allowable values of the associated instance

data item to valid representations of the lexical space of the datatype, including enforcing of any constraining facets.

Example Schema Syntax: declaring a datatype based on an `xsd:string` plus additional constraining facet would be accomplished as follows:

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

5.3.2 Closed Enumeration

Often it is necessary to restrict the allowable values of the associated instance data item to a closed list of alternatives. Also under consideration is a method to obtain a list at runtime, for example, from an XPath node-set.

Example Schema Syntax: declaring a datatype allowing enumerated values of an `xsd:string` would be accomplished as follows:

```
<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Mastercard"/>
    <xsd:enumeration value="Diner's Club"/>
    <xsd:enumeration value="American Express"/>
  </xsd:restriction>
</xsd:simpleType>
```

5.3.3 Open Enumeration

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

Example Schema Syntax: declaring an open enumeration is possible through a combination of union and enumeration features, as follows:

```
<xsd:simpleType>
  <xsd:union memberTypes="xsd:string">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Mastercard"/>
        <xsd:enumeration value="Diner's Club"/>
        <xsd:enumeration value="American Express"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

5.3.4 Union

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

Example Schema Syntax: declaring a datatype allowing either a `creditCardType` or `bonusProgramType` value would be as follows:

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

5.3.5 Multiple Selection

Some form controls, such as `selectMany`, have the notion of supporting more than one simpleType value at any given time. This corresponds with Schema list datatypes.

Example Schema Syntax: declaring a list-derived datatype would be as follows:

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

5.3.6 Repeating Line Item

It is common for certain types of forms, such as order forms, to contain repeating structures, typically line items. If each individual structure were represented as a Schema complexType, a line item group would be analogous to the `sequence` construct.

Example Schema Syntax: a datatype allowing a sequence of child elements would be declared as follows:

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="child" type="xform:string" minOccurs="0" maxOccu
  </xsd:sequence>
</xsd:complexType>
```

Chapter **8 XForms User Interface** contains details on representing this with XForms User Interface form controls, as well as details for how this relates to the instance data in chapter **11 Processing Model**.

5.3.7 Alternate Representation

In some forms, alternate representations might be necessary for underlying instance data structures.

Example Schema Syntax: a Schema `choice` element is roughly analogous to this, although XForms uses a more dynamic version. Examples of this are found in **8 XForms User Interface**

6 XPath Expressions in XForms

XPath is used within XForms to address instance data, as well as to perform basic operations, such as declaratively stating when a form control needs to be filled out, or defining a computation over other values such as unit prices, quantities, discounts, and tax and shipping costs. This chapter describes how XForms uses XPath, and additional XForms functions for use in forms.

6.1 Datatypes

In general, XPath uses a smaller set of datatypes than XML Schema. XForms allows arbitrary Schema datatypes, including those defined in **4 Datatypes**, while XPath datatypes are limited to `boolean`, `string`, `number`, and `node-set`. (For completeness, XPath additionally has external objects and result tree fragments, but there is no special treatment for these types in the XForms specification.)

> **Note:**
>
> Resource-limited XForms Processors may define implementation limits on the maximum size of a `node-set`.

The XForms specification is defined such that it is always clear whether XPath or XML Schema datatypes are used within a particular context. Binding expressions and computed expressions always use XPath datatypes, while everything else uses XML Schema datatypes.

> **Note:**
>
> A future version of XForms is expected to use XPath 2.0, which includes support for XML Schema datatypes, which will make the above distinction moot.

6.2 Instance Data

Every form has a current state, representing the values entered at any particular point in time. Within XForms, for each `xform` element, the XForms Processor must behave as if it internally maintains an XML data structure modeled as a tree to represent the state of the form. This data structure is called instance data and conforms to the XPath Data Model [XPath 1.0]. Additionally, each node in the tree contains a boolean "dirty" flag, which is referenced elsewhere by the XForms Processing Model. In this context, "dirty" indicates that the data value might need to be refreshed in the presentation.

Elements and attributes in the instance data may have namespace information associated with them, as defined in the XPath Data Model. Unless otherwise specified, all instance data elements and attributes are unqualified. Instance data

elements and attributes may not belong to the XForms namespace.

Issue (issue-instance-data-access):

Should there be specified a DOM form of access (perhaps a document fragment), that maps to the instance data? If so, should it be read-only or read-write access? What are possible security implications here?

The rules for defining the root and context nodes of the instance data are found in in the following section.

6.3 Evaluation Context

Applied to XForms, XPath references abstract instance data (using the "path" portion of XPath), instead of a discrete XML document. This reference is called a binding expression in this specification.

The following context is used for evaluating all XPath expressions in XForms:

1. The context node for **outermost** binding elements (such as XForms UI elements) is the XPath root (`/`). A "**binding element**" is any element that is explicitly allowed to have an `xform:ref` attribute. An XForms element is "**outermost**" when the node-set returned by the XPath expression `ancestor::*` includes no **binding element** nodes.

   Note:

   The contents of the instance data below the XPath root node (`/`) are dependent on how the instance data was constructed, which is defined in **11.4.2 Instance Data Construction**.
2. The context node for non-**outermost** binding elements is determined by evaluating 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 size and position are both exactly 1.
4. No variable bindings are in place.
5. The available function library is defined below.
6. Any namespace declarations in scope for the attribute that defines the expression are applied to the expression.

Example:

```
<repeat ref="element1/foo/bar">
  <selectOne ref="element2" ... />
  <selectOne ref="@attr" ... />
</repeat>
```

In this example, the `repeat` has a binding expression of `element1/foo/bar`. According to the rules above, this outermost element would have a context node of `/`, which is the root of the instance data, or the parent to the `element1` element.

Both of the `selectOne`s then inherit a context node from their parent, the context node being `/element1/foo/bar`. Based on this, the `selectOne` binding expressions evaluate respectively to `/element1/foo/bar/element2` and `/element1/foo/bar/@attr`. Matching instance data follows:

```
<element1>
  <foo>
    <bar attr="xyz">
      <element2>xyz</element2>
    </bar>
  </foo>
</element1>
```

6.4 Canonical Binding Expressions

As with XPath, it is possible to construct many different binding expressions that end up returning the same node-set. That said, it is often useful to express a binding expression in a standard, compact representation, defined as a canonical binding expression.

Canonical binding expressions are represented as an `AbsoluteLocationPath` as defined in [XPath 1.0]. Additionally, canonical binding expressions use only default abbreviated axis-specifiers (for elements) or the '@' abbreviation (for attributes). Examples:

- (canonical) `/a/b/c`
- (canonical) `/a/b/@c`
- (non-canonical) `a/b/c` (not an absolute path)
- (non-canonical) `child::a/child::b/child::c`
- (non-canonical) `/a/b/c/d/ancestor::c`

6.5 Forms Core Function Library

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

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

> **Issue (xpath-core-lib):**
>
> Further input is required on the ability for resource-constrained devices to implement the complete XPath Core Function Library.

6.5.1 Number Methods

number **average**(*node-set*)

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

computed with `count()`.

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

The `min` function returns the minimum value, for each node in the argument node-set, of the result of converting the string-values of the node to a number. "Minimum" is determined with the `<` operator.

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

The `max` function returns the maximum value, for each node in the argument node-set, of the result of converting the string-values of the node to a number. "Maximum" is determined with the `<` operator.

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

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

> **Note:**
>
> The following core functions are defined within [XPath 1.0] - `number(), sum(), floor(), ceiling(), and round()`

> **Note:**
>
> The following useful numeric and boolean operators are defined within [XPath 1.0] - "`+`", "`-`", "`*`", "`div`", "`mod`", unary "`-`" ,"`=`", "`!=`", "`<`", "`>`", "`<=`", "`>=`", "`or`", "`and`".

6.5.2 String Methods

string **now**()

The `now` function returns the current system time as a string value, in the canonical format defined within the XForms specification. If local time zone information is available, it is included in the string.

> **Note:**
>
> Note: the following are defined within [XPath 1.0] - `string(), concat(), starts-with(), contains(), substring-before(), substring-after(), substring(), string-length(), normalize-space(), and translate()`.

6.5.3 Miscellaneous Methods

boolean **submit**()

The `submit` function immediately submits the instance data bound to the node that contains the expression by triggering an `xforms-submit` event.

boolean **reset**()

The `reset` function immediately resets the instance data bound to the node that contains the expression by triggering an `xforms-reset` event.

string **xforms-property**(*string*)

The `xforms-property` function accesses the XForms Property (defined in **11 Processing Model**) named by the string parameter, and returns the value of the property.

6.6 Extensibility

> **Issue (user-extensions):**
>
> This section will be expanded in future revisions, to cover extension functions and methods for calling out to script, perhaps something along the lines of `element-available` and `function-available`.

7 Form Controls

7.1 Introduction

XForms User Interface controls, also called form controls, are declared using markup elements, and their behavior refined via markup attributes. This markup may be further decorated with style properties that can be set using CSS stylesheets to deliver a customized look and feel. Form controls defined here are bound to the underlying instance data using the binding attributes as defined in the chapter **9 Binding**.

The XForms form controls enable accessibility by taking a uniform approach to such features as captions, help text, tabbing and keyboard shortcuts. Internationalization issues are being addressed in conjunction with the Internationalization Working Group and are addressed by following the same design principles as within the rest of XHTML. All form controls defined here are suitable for implementation as Aural CSS (ACSS) form controls.

Several XForms form controls are of a general class that represent the concept of selecting from available choices. Such selection controls can be characterized along a presentational dimension that is completely orthogonal to the functional distinction. Distinguishing the *presentational* from the *functional* dimension allows the expression of the *meaning* of a particular form control--see [AUI97] for a definition of such high-level user interaction primitives.

This chapter includes non-normative graphical examples of many form controls. The CSS Working Group is providing assistance with creating default CSS rules for producing visual renderings of standard XForms form controls. This specification will also include non-normative rules for how these same controls might be rendered to alternative access modalities.

Issue (control-names):

All form control names listed here should be considered advisory until further consensus is reached in the Working Group.

For each form control, the following aspects will be defined:

Description
Examples
Data Binding Restrictions
Implementation Hints
XML Representation

The form controls defined here use common attributes and elements that are defined later in this chapter (**7.13 Common Markup**).

7.2 textbox

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

Examples:

```
<textbox ref="order/shipTo/street" style="width:xx; height:xx">
 <caption>Street<caption>
 <help>Please enter the number and street name</help>
</textbox>
```

In the above, CSS style attributes `height` and `width` 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: The entered value of the form control (after processing as described in **11 Processing Model**) is treated as a lexical value. A datatype bound to this form control will be treated as a restriction upon the allowed entered value.

Implementation Hints: Implementations may represent this form control with more than one native user interface control, for example a form control that appears to be three separate smaller entry fields for "day", "month", and "year" for a date datatype. Further, for date datatypes, a calendar system for data entry may be used, including non-Gregorian calendar systems. For numeric datatypes, additional features might include spin buttons or other conveniences. When bound

to a datatype that accepts newline characters, this form control should accept multi-line input.

Example: XML Representation: <textbox>

```
<textbox
  (common attributes)
>
  <!-- caption, help?, hint?, onevent? -->
</textbox>
```

common attributes defined in **7.13.1 Common Attributes**

7.3 secret

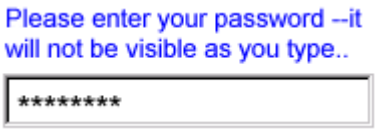
Description: This form control is used for obtaining information that is considered sensitive, and thus not echoed to a visual or aural display as it is being entered, e.g., password entry.

Example:

```
<secret ref="/login/password" style="width:xx; height:xx">
  <caption>Please enter your password --it will not be visible as you t
</secret>
```

In the above, CSS style attributes `height` and `width` 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 this form control as follows:



Data Binding Restrictions: Identical to `textbox`.

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

Example: XML Representation <secret>

```
<secret
  (common attributes)
>
  <!-- caption, help?, hint?, onevent? -->
</secret>
```

common attributes defined in **7.13.1 Common Attributes**

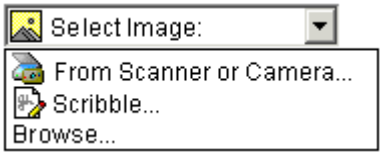
7.4 uploadMedia

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

Example:

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

A graphical browser might render this form control as follows:



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

Implementation Hints:

- Implementations with a file system SHOULD support "file upload"--selecting a specific file, for all mediaTypes. The types of files presented by default MUST reflect the mediaType specified in the XForms Model, for example defaulting to only audio file types in the file dialog when the mediaType is "audio/*". In XForms 1.0, there is a 1:1 binding between a uploadMedia form control and one of the `xform:binary` datatypes, although that single file may be compound (e.g. application/zip).
- Implementations with specific pen/digitizer hardware SHOULD (and implementations with other pointing devices MAY) support "scribble"--allowing in-place creation of pen-based PNG image data, when the mediaType is "image/*" or "image/png". Other mediaTypes besides image/png MAY share this input method.

  Note:

  Commonly, people have trouble drawing recognizable figures or signatures with a conventional pointing device like a mouse or track ball. Hence, a reasonable implementation of XForms might not want this

feature, hence the "MAY" here for generic pointing devices

- Implementations with specific audio recording capabilities SHOULD support *record*--in-place recording of an audio clip, when the mediaType is "audio/*" or "audio/basic". Other mediaTypes besides "audio/basic" MAY share this input method.
- Implementations with a digital camera/scanner interface SHOULD support *send image*--in-place upload of images from an attached device, when the mediaType is "image/*" or "image/jpeg". Other mediaTypes besides "image/jpg" MAY share this input method.
- Implementations with video recording capability SHOULD provide a "record" option for video/* mediaTypes.
- Implementations with 3d capabilities SHOULD provide a 3d interface option for model/* mediaTypes.
- 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.

Example: XML Representation: <uploadMedia>

```
<uploadMedia
  (common attributes)
  mediaType = list of content types
>
  <!-- caption, help?, hint?, onevent? -->
</uploadMedia>
```

common attributes defined in **7.13.1 Common Attributes**
mediaType = list of media types - list of suggested media types, used by the XForms Processor to determine which input methods apply.

7.5 selectOne

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

Typically, a stylesheet would be used to determine the exact appearance of form controls, though a means is provided to make a concrete selection through an attribute. The value of the attribute consists of one of the following values, each of which may have a platform-specific behavior:

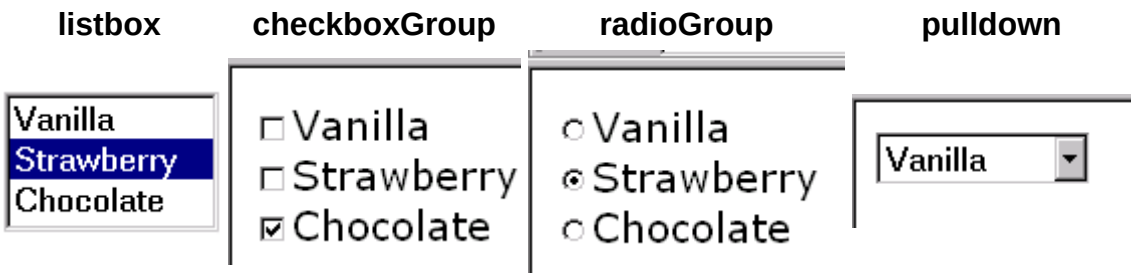
radioGroup
checkboxGroup
pulldown
listbox
comboGroup

Example:

```
<selectOne ref="icecream/flavor">
  <caption>Flavor</caption>
  <choices>
    <item value="vanilla">Vanilla</item>
    <item value="strawberry">Strawberry</item>
    <item value="chocolate">Chocolate</item>
  </choices>
</selectOne>
```

In the above example, selecting one of the choices will result in the associated `value` given by attribute `value` on the selected item being set in the underlying data instance at the location `icecream/flavor`. The `value`s given in the user interface shown above may be used in constructing a default schema if no schema is provided by the XForms author.

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



Data Binding Restrictions: This form control will select the lexical value from the `value` attribute (or in the absence of such an attribute, the text content of the `item` element) of the single item selected. If the datatype bound to this form control does not permit the selected value (for instance a datatype of `xsd:decimal` with an attribute `value="abc"`), the form control with that selection will be perpetually considered invalid and it will not be possible to submit the form. Authors are encouraged to avoid this situation.

If the datatype bound to this form control includes a non-enumerated value space (for instance `xsd:string`, or `xsd:string` as part of a union), or if the "comboGroup" UI hint is specified, the form control then should allow free data entry, as described in **7.2 textbox**, in addition to the behavior defined here.

Issue (items-specified-elsewhere):

Yet to be decided is allowing display and/or storage values located elsewhere to be retrieved from a binding expression.

Some user interface combinations may allow a state of zero selected items, in which case the lexical value of a zero-length string is selected.

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

Example: XML Representation: <selectOne>

```
<selectOne
  (common attributes)
  selectUI = ("radioGroup" | "checkboxGroup" | "pulldown" | "listbox" |
>
  <!-- caption, help?, hint?, onevent?, choices* -->
</selectOne>
```

common attributes defined in **7.13.1 Common Attributes**
selectUI = ("radioGroup" | "checkboxGroup" | "pulldown" | "listbox" | "comboGroup") - appearance override

7.6 selectMany

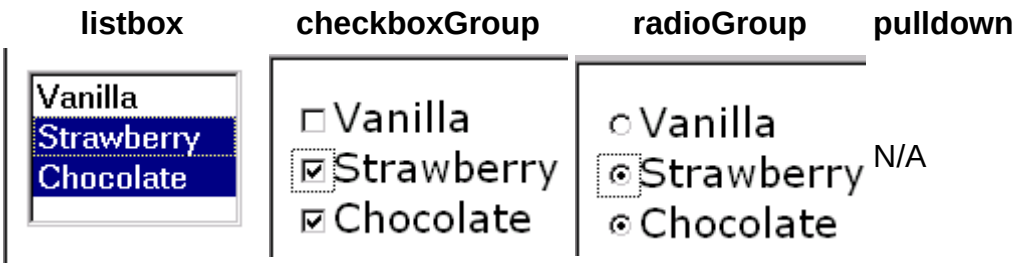
Description: This form control allows the user to make multiple selections from multiple choices.

Example:

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

In the above example, more than one flavor can be selected, populating the instance data with multiple selections.

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



Data Binding Restrictions: When zero or one items are selected, this form control behaves exactly like `selectOne` with regard to the lexical value that is selected. When multiple items are selected, the lexical value is a space-separated list of the selected values. The datatype bound to this form control must be capable of supporting this format, typically a Schema list type. Cases where each of the multiple selections appear in the instance data attached to a separate element are handled through the `repeat` construction (**8.2 Repeating Structures**).

Note:

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

For instance, the following incorrect item declaration:

```
<item value="United States of America">USA</item>
```

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

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

Example: XML Representation: <selectMany>

```
<selectMany
  (common attributes)
  selectUI = ("radioGroup" | "checkboxGroup" | "pulldown" | "listbox" |
>
  <!-- caption, help?, hint?, onevent?, choices* -->
</selectMany>
```

common attributes defined in **7.13.1 Common Attributes**
selectUI = ("radioGroup" | "checkboxGroup" | "pulldown" | "listbox" | "comboGroup") - appearance override

7.7 selectBoolean

Description: This form control represents an on/off or true/false or yes/no (or similar) choice.

Example:

```
<selectBoolean ref="questionnaire/married">
  <caption>Are you married?</caption>
  <help>We need this to determine your tax allowance</help>
  <choices>
    <item value="true">Yes</item>
    <item value="false">No</item>
  </choices>
</selectBoolean>
```

Data Binding Restrictions: This form control produces only two possible lexical

values: `true` or `false`. To be considered valid, the datatype bound to this form control (typically `xform:boolean`) must be able to accept these two lexical values.

Note:

Scenarios where the desired lexical value is anything other than 'true'/'false' are not suitable for the `selectBoolean` form control.

For example, if the values placed into the instance data were required to be either "male" or "female", the `selectOne` form control should be used instead.

Implementation Hints: Visual implementations would typically render this as a checkbox. In some cases, like the above example or in aural environments, it may be helpful to provide labels for the respective choices. This is accomplished through the `choices` mechanism, similar to the other `select...` form controls.

Example: XML Representation: <selectBoolean>

```
<selectBoolean
  (common attributes )
  selectUI = ("radioGroup" | "checkboxGroup" | "pulldown" | "listbox" |
>
  <!-- caption, help?, hint?, onevent?, choices* -->
</selectBoolean>
```

common attributes defined in **7.13.1 Common Attributes**
selectUI = (TBD)

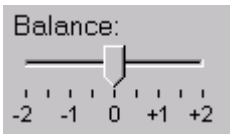
7.8 range

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

Example:

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

A graphical browser might render this as follows:



Data Binding Restrictions: Only datatypes which represent a continuous range where it is possible to express a difference value can be bound to this form control. (For instance, `xform:decimal` would be fine, while `xform:string` or `xform:.Binary` would not). In terms of Schema datatypes, the datatype must be either 1) have a total order relationship, or 2) an overall partial order relationship,

but totally ordered within the range specified between the `start` and `end` attributes.

Issue (enum-range):

Should an enumeration be allowed to bind to this form control? If yes, how should it be ordered?

Implementation Hints: In graphical environments, this form control would typically be rendered as a "slider" or "volume 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 like speech, e.g., when using an accessibility aid. Thus, an Aural CSS enabled user agent might speak a prompt of the form *Please pick a date in the range January 1, 2001 through December 31, 2001.*

Example: XML Representation: <range>

```
<range
  (common attributes)
  start = datavalue
  end = datavalue
  stepSize = datavalue-difference
>
  <!-- caption, help?, hint?, onevent? -->
</range>
```

common attributes defined in **7.13.1 Common Attributes**
start = datavalue - Lexical starting bound for the range, of the same datatype bound to the form control
end = datavalue - Lexical ending bound for the range, of the same datatype bound to the form control
stepSize = datatype-difference - Prefered step-size to use for incrementing or decrementing the value within the form control, of a datatype that can express the difference between two values of the datatype bound to the form control

7.9 button

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

Example:

```
<button>
Example unavailable at time of publication
</button>
```

Data Binding Restrictions:

Note:

Binding a model item has no direct effect on a button, but provides a context for any event handlers that are attached.

Implementation Hints: Graphical implementations would typically render this form control as a push-button.

Example: XML Representation: `<button>`

```
<button
  (common attributes)
>
  <!-- caption, help?, hint?, onevent? -->
</button>
```

common attributes defined in **7.13.1 Common Attributes**

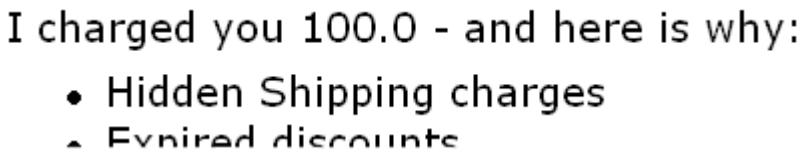
7.10 output

Description: This form control renders a value from the instance data, but provides no means for entering or changing data. This form control may be used in a `caption`, for instance, when authors want to say: "I charged you *value* - and here is why.

Example:

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

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



Data Binding Restrictions: The lexical value of the datatype bound to this form control is displayed, after processing as described in **11 Processing Model**.

Implementation Hints: An audio browser might apply properties to this form control to *aurally highlight* the displayed value to provide audio formatted output.

Example: XML Representation: `<output>`

```
<output
 id = xsd:ID
 ref = binding-expression
 xform = instance data selector
 format = formatting-expression
>
  <!-- empty content -->
</output>
```

id = xsd:ID - Optional unique identifier used for linking.
ref = binding-expression - Binding expression
xform = xsd:IDREF - Optional instance data selector. Details in the chapter **9 Binding**.
format = formatting-expression - Optional format specifier

Editorial note	
We need to decide on how we define attribute `format` on form control `output`. The functionality needed is similar to what functions like `printf` typically take.	

7.11 submit

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

Example:

```
<submit xform="timecard">
  <caption>Submit</caption>
</submit>
```

Implementation Hints: The default handling for this controls is equivalent to the `submit()` method.

Example: XML Representation: `<submit>`

```
<submit
  (common attributes)
>
  <!-- caption, help?, hint?, onevent? -->
</submit>
```

common attributes defined in **7.13.1 Common Attributes**

7.12 reset

Description: This form control resets to the initial values all or part of the instance data to which it is bound.

Example:

```
<reset ref="/tcard/data" xform="timecard">
  <caption>Reset totals</caption>
</reset>
```

Implementation Hints: The default handling for this controls is equivalent to the `reset()` method.

Example: XML Representation: `<reset>`

```
<reset
  (common attributes)
>
  <!-- caption, help?, hint?, onevent? -->
</reset>
```

common attributes defined in **7.13.1 Common Attributes**

7.13 Common Markup

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

7.13.1 Common Attributes

Issue (XHTML-attrs):

XHTML defines two attributes on element `html:form`--`accept` and `accept-charset`. Additionally, attribute `accept-charset` also appears on element `html:input`. We need to bring the equivalent to these into the XForms specification.

Example: XML Representation: Common Attributes

```
xmlns = xsd:anyURI
xml:lang = xsd:language
id = xsd:ID
class = space separated list of classes
style = associated style info
ref = binding-expression
xform = xsd:IDREF
navIndex = xsd:nonNegativeInteger : 0
accessKey = xsd:token
```

xmlns = xsd:anyURI - Optional standard XML attribute for identifying an XML namespace.

xml:lang = xsd:language - Optional standard XML attribute to specify a human language for this element.
id = xsd:ID - Optional unique identifier used for linking.
class = space separated list of classes - Optional selector for a style rule.
style = associated style info - Optional inline style specification.
ref = binding-expression - Binding expression. Details in the chapter **9 Binding**.
xform = xsd:IDREF - Optional instance data selector. Details in the chapter **9 Binding**.
navIndex = xsd:nonNegativeInteger : 0 - 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 **11 Processing Model**.
accessKey = xsd:string - Optional attribute defines a shortcut for moving the input focus directly to a particular form control. The value of this is typically a single character which when pressed together with a platform specific modifier key (e.g. the *alt* key) results in the focus being set to this form control.

CSS properties for controlling the look and feel of XForms form controls are being defined in conjunction with the CSS Working Group. This version of the XForms working draft defines the XForms form controls independent of visual presentation. Additionally, sample default visual presentations are shown for each form control defined in this Working Draft. The CSS Working Group has agreed to help us develop a default CSS stylesheet capable of producing the sample default renderings illustrated in this working draft. The results of the above will be used to document the use of CSS properties within XForms user interface elements for the final version of the XForms specification.

7.13.2 Common Child Elements

Child elements `caption`, `help` and `hint` detailed below provide the ability to attach human-readable metadata to XForms form controls.

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

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

7.13.2.1 caption

The required element `caption` labels the containing form control with a descriptive

label. Additionally, the caption makes it possible for someone who can't see the form control to obtain a short description while navigating between form controls.

Example: XML Representation: <caption>

```
<caption
  (common attributes)
>
  <!-- mixed content -->
</caption>
```

common attributes defined in **7.13.1 Common Attributes**

An accessibility aid would typically speak the metadata encapsulated in element `caption` when the containing form control gets focus.

7.13.2.2 help

The optional element `help` provides a longer description that will help users understand how to fill out this form control. The `help` text will be shown only on request.

Example: XML Representation: <help>

```
<help
  (common attributes)
>
  <!-- mixed content -->
</help>
```

common attributes defined in **7.13.1 Common Attributes**

A graphical browser might render help as follows:

An accessibility aid might speak this information upon request.

7.13.2.3 hint

The optional element `hint` provides a short hint for the user, typically represented

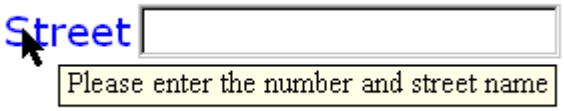
as a tooltip by graphical user agents. The tooltip text will normally be shown when the user remains on the form control for more than a certain length of time. Accessibility aids might render such tooltips using speech. This element is optional, and its content model is mixed.

Example: XML Representation: <hint>

```
<hint
  (common attributes)
>
  <!-- mixed content -->
</hint>
```

common attributes defined in **7.13.1 Common Attributes**

A graphical browser might render hints as follows:



7.13.2.4 onevent

This element can be used to bind event handlers to form controls. It is defined in [XHTML Events]. Details on XForms events can be found in the chapter **11 Processing Model**.

Example: XML Representation: <onevent>

```
<onevent
  (attributes defined in XHTML Events)
>
  <!-- Action handlers -->
</onevent>
```

Element `onevent` is defined in the XHTML Events module. It declares an event listener by specifying the event to handle and the event handler to invoke.

7.13.2.5 item

This element is used within list form controls to represent a single item of the list.

Issue (onevent-for-item):

Should onevent handlers be allowed here for individual items in a list?

Example: XML Representation: <item>

```
<item
  value = lexical-representation
>
  <!-- #PCDATA -->
</item>
```

value = lexical-representation - the "storage value" for the item, to be placed in the instance data when this item is chosen.

7.13.2.6 choices

This element is used within list form controls to group items. This provides the same functionality as element `optgroup` in HTML 4.0.

Example: XML Representation: `<choices>`

```
<choices>
  <!-- (item | choices)+ -->
</choices>
```

8 XForms User Interface

8.1 Conditional Constructs For Dynamic User Interfaces

The XForms User Interface allows the authoring of dynamic user interfaces, i.e., user interfaces that vary based on the current state of the instance data being populated. As an example, portions of a questionnaire pertaining to the user's automobile may become *relevant* only if the user has answered in the affirmative to the question 'Do you own a car?'. Another use case for such dynamic user interfaces is when the underlying XForms Model contains conditional structures.

8.1.1 switch

Editorial note: Raman	
Please do not attach semantics to the element name to prejudge the design.	

This element contains one or more `case` elements. Each `case` has an `id` attribute that is used within event handlers appearing within the form controls to activate or deactivate that portion of the user interface created by the markup contained within that `case` element.

```
<switch id="01" default="initial">
  <case id="us">
    <group>
      <caption>Please Specify a US Shipping Address.</caption>
    </group>
  </case>
  <case id="uk">
    <group >
      <caption>Please specify a UK shipping address.</caption>
    </group>
  </case>
  <case id="initial">
    <group >
      ...
    </group>
  </case>
</switch>
```

The above will result in the portion of the user interface contained within the *default case* being initially displayed. Next, we define an XForms event handler `toggle` below for activating and deactivating different portions of this conditional construct.

Example: XML Representation: Action <`toggle`>

```
<toggle switch="switchID" case="caseID" />
```

Attribute `switch` identifies the switch constructor to be affected.
Attribute `case` identifies the portion of the `switch` construct to activate.

In addition to event handler `toggle`, event handler `scroll` might be used within construct `switch` to cycle through the various contained `case` clauses.

8.2 Repeating Structures

The XForms Model allows the definition of repeating structures such as multiple items being purchased within a purchase order. When defining the XForms Model, such higher-level aggregations are constructed out of basic building blocks; similarly, here we define higher-level user interface constructs that build on the form controls defined previously, that can bind to data structures such as lists and collections.

8.2.1 Design Rationale

The basic XForms form controls defined so far permit populating data instances conforming to Schema simpleTypes. These form controls can be aggregated using `group`(see **8.4 Layout in XForms**) to form higher-level user interface form controls. As an extension to this, the user interface markup for repeating structures only adds encapsulation metadata about the collection being populated, as well as the necessary controls needed for adding, deleting and navigating the items in the repeating structure. Finally, this section also defines relevant portions

of the processing model that track the item that is *current* in a repeat structure..

Element `repeat` encapsulates the following pieces of metadata:

- *Binding expression* specifying the collection to be populated by the contained XForms form controls. This binding expression returns a *node-set* that represents the collection of items over which this `repeat` will iterate.

  Issue (nodeset-vs-ref):

  Question: should we call this binding attribute `node-set` as opposed to `ref` to highlight that we use the returned node-set rather than the single desired node as in most other usages of the `ref` attribute? Note that submit and reset also use full node-set processing.
- *Starting index* of the first element from the collection to be displayed to the user.
- *Number of elements* from the collection to be displayed to the user.

8.2.2 Special Event Handlers For Element `repeat`

We define XForms-specific event handlers for use within element `repeat`. These event handlers will appear within element `onevent` to specify the event handler to trigger when an event is received by the containing event listener. In the examples below, we will assume event `dom-click`; however, note that this specification does not normatively define the mapping between physical events and specific behaviors.

Construct `repeat` introduces the notion of a *cursor* that is maintained by the XForms processing model for each repeating structure. Thus, each `repeat` has its own a conceptual *cursor*. Event handlers are provided for querying and updating the position of this cursor; this cursor position in turn is used for determining the behavior of `insert` and `delete`. Thus, these event handlers can be thought of as the markup equivalent of the additions to the XForms DOM introduced by construct `repeat`.

1. `setRepeatCursor` for marking an item as being *current*.

   Example: XML Representation:Action <setRepeatCursor>

   ```
   <setRepeatCursor repeat="repeatId" cursor="itemID"/>
   ```

   Attribute `repeat` identifies the repeat constructor whose *cursor* is to be updated.
   Attribute `cursor` identifies the *cursor* position. It should be a binding expression that evaluates to an element of the node set being iterated over by this repeat construct.

2. `getRepeatCursor` for accessing position of the repeat cursor.

   Example: XML Representation:Action <getRepeatCursor>

```
<getRepeatCursor repeat="repeatId" />
```

Attribute `repeat` identifies the repeat constructor whose *cursor* position is to be obtained.

3. `scroll` for changing item that is *current*.

Example: XML Representation:Action <scroll>

```
<scroll repeat="repeatId" step="n" />
```

Attribute `repeat` identifies the repeat constructor whose *cursor* position is to be updated.
Attribute `step` is an integer value that specifies the amount by which the list is scrolled. Negative values may be used to scroll the list in the reverse direction.

Issue (additional-actions):

We may add special values for scrolling to the top or bottom of the list, as well as other needed actions.

4. `insert` for inserting an item at the current cursor position into the repeating structure. This causes the instantiation of the necessary user interface for populating a new entry in the underlying collection. It also updates the instance data by instantiating the necessary nodes.

Example: XML Representation:Action <insert>

```
<insert repeat="repeatId" />
```

Attribute `repeat` identifies the repeat constructor being affected, and defaults to the `repeat` construct within which this event handler appears.

5. `delete` for deleting an item from the repeating structure. This causes the destruction of the necessary user interface for populating the entry at the current cursor position in the underlying collection. It also updates the instance data by destroying the necessary nodes.

Example: XML Representation:Action <delete>

```
<delete repeat="repeatId" />
```

Attribute `repeat` identifies the repeat constructor being affected,

and defaults to the `repeat` construct within which this event handler appears.

The event handlers enumerated above may be used within event listeners attached to `button` elements placed inside the body of construct `repeat`. In combination with the facilities provided by element `group` and CSS layout, this allows document authors to customize the positioning of controls for standard actions such as insert or delete.

8.2.3 repeat

Element `repeat` represents a repeating homogenous collection, in other words, a collection consisting entirely of like form controls.

```
<repeat ref="bindingExpr" startIndex="si" number="n">
  <caption>Shopping Cart</caption>
  <!-- XForms UI for one element of the collection -->
  <!-- controls for adding, deleting and navigating entries -->
</repeat>
```

Notice that the underlying schema for the collection being populated would typically have defined attributes `minOccurs` and `maxOccurs`; these values will in turn determine if the user agent displays appropriate UI controls for the user to add or delete entries in the collection. Event listeners within element `repeat` control UI aspects such as scrolling and insertion. This specification will not specify normative means for displaying such add and delete controls; as with the rest of the XForms User Interface design, we leave these to be flexible while showing a default presentation.

Example: XML Representation: `<repeat>`

```
<repeat
  (common attributes)
  startIndex = xsd:positiveInteger : 1
  number = xsd:nonNegativeInteger
>
  <!-- caption, help?, hint?, onevent?, ( any form control )* -->
</repeat>
```

common attributes defined in **7.13.1 Common Attributes**
startIndex = xsd:positiveInteger : 1 - 1-based hint to the XForms Processor as to which starting element from the collection to display.
number = xsd:nonNegativeInteger - hint to the XForms Processor as to how many elements from the collection to display.
any form control - any form control defined in **7 Form Controls**

8.2.4 Design Consequences

This subsection enumerates the design consequences and is for members of the Working Group to evaluate the above design.

- Binding expression is placed on element `repeat` and should refer to the collection being populated, not to an individual item of the collection. Thus, it would be `items/item` in a purchase order, where element `items` contains one or more `item` children.
- The form controls appearing inside element `repeat` needs to be suitable for populating individual items of the collection. Thus, to continue the purchase order example, the contained XForms form controls would need to be suitable for populating a data instance conforming to `item`.
- A simple but powerful consequence of the above is that if the XForms Model specifies nested collections, then we can create a corresponding user interface by *nesting* `repeat` elements. As an example, consider the XForms Model for a hypothetical purchase order that contains element `items` for holding collection of `item` elements. Further, assume that element `item` comprises of two atomic fields `catalogNumber`, `partNumber` and a collection `colors` which in turn holds one or more `color` elements. The user interface for populating this data instance would use nested `repeat` elements.
- 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. But as a consequence, the user interface cannot bind to an element from the underlying collection *out of sequence*. Notice that this is not a serious drawback because the use interface layer can always populate a specific member of a collection without using element `repeat`.

8.3 Reusable Form Controls

8.3.1 Creating User Interface Templates

User Interface templates allow the creation of reusable user interface components. Just as we can define data types and structures that can be reused within the XForms Model, reusable user interface components allow us to design complex user interfaces using the basic building blocks described in the previous section, and then reuse these components in multiple situations. As with any component framework, this has two basic requirements:

- Components need to declare what aspects of the component are parameterizable by the caller.
- The caller needs to be able to override the default values of the parameters declared in the component.

Here, we describe such a component framework along with sample markup. For this example, assume that `USShippingAddress` is a reusable data type that is used in multiple places in the XForms Model, e.g. the user will be asked for a `billingAddress` and `shippingAddress`--both of type `USShippingAddress`.

First, we show a simple example that is designed to bind an XForms form control to a model item of type `address` with no attention to making the component reusable.

```
<group>
  <textbox ref="address/street">
    <caption>Please enter your street address</caption>
  </textbox>
  <textbox ref="address/zip">
    <caption>Zip Code</caption>
  </textbox>
</group>
```

Next, we prepare the above fragment to become a reusable component that could be used for obtaining both the shipping and billing address. To do this, we need to parameterize those portions of the *component* that the caller will wish to modify.

```
<uiTemplate id="AddressWidget">
  <param name="streetPrompt"/>
  <param name="zipPrompt"/>
  <param name="border" value="line"/>
  <group border="$border">
    <textbox ref="address/street">
      <caption><value-of name="streetPrompt"/></caption>
    </textbox>
    <textbox ref="address/zip">
      <caption><value-of name="zipPrompt"/></caption>
    </textbox>
  </group>
</uiTemplate>
```

Note that the markup shown above does not create a user interface; user interface is created by explicitly instantiating the component via element `useUITemplate` described next.

Next, we use this component to instantiate the user interface for obtaining the shipping and billing address.

```
<useUITemplate ref="myAddress" component="AddressWidget">
  <withParam name="streetPrompt">Shipping Street Address</withParam>
  <withParam name="zipPrompt">Zip Code for shipping state</withParam>
  <withParam name="border">dotted</withParam>
</useUITemplate>
```

The reusable component is instantiated by element `useUITemplate`; parameter values are specified by the contained `withParam` elements. Attribute `xform` sets the binding context relative to which binding expressions within the instantiated template get evaluated.

8.3.2 DTD For uiTemplate And useUITemplate

TODO: convert to ’XML Representation’ consistent with rest of spec.

```
<!--       param element for use in uiTemplate -->
<!ELEMENT  param      EMPTY>
<!--       name       Name of parameter being declared          -->
<!ATTLIST  param
  name     CDATA      #REQUIRED>
<!--       Defines a reusable user interface template -->
<!ELEMENT  uiTemplate   (param*, %FormControls;+)>
<!--       id         Unique ID for use when instantiating this templat
<!ATTLIST  uiTemplate
  id       ID         #REQUIRED>
<!--       Used to pass parameter values when instantiating uiTemplate
<!ELEMENT  withParam  #MIXED>
<!--       name       Name of parameter whose value is being specified
<!ATTLIST  withParam
name       CDATA      #REQUIRED>
```

8.4 Layout in XForms

The `group` element is used as a container for defining a hierarchy of form controls. Groups lay out their children using a constraint-based system that supports both relative flexible sizing and intrinsic sizing. Groups can be nested to create complex hierarchies. Using CSS, an inline box can be specified using the `display` property with a value of `inline`. A block-level group can be specified with a value of `block`, which is the default.

Example: XML Representation: <`group`>

```
<group>
  <!-- all form controls & mixed content -->
</group>
```

All form controls defined so far are treated as inline text for purposes of XHTML processing. XForms visual clients are expected to use a a box layout model as defined by CSS for determining the overall layout of the XForms visual interface. Grouping of form controls using element `group` also provides semantics about the relationship amongst user interface controls; such knowledge can be useful in delivering a coherent UI to small devices--e.g., if the user interface needs to be split up amongst several screens, controls appearing inside the same group would typically be rendered on the same screen.

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

Issue (group-binding):

t is an open issue whether the binding attribute `xform` is allowed on element `group`. It might make authoring easier but makes implementations significantly harder. Note that this note is here because at one point in our design we said that controls inside a `group` could use relative XPath expressions with the

context being set by the containing group.

8.4.1 Orientation and Direction

Containers typically have an orientation and a direction. The `box-orient` CSS property specifies the orientation of the group. It has values of `horizontal`, `vertical`, or `inherit`. A horizontal group lays out its children in a horizontal line, and a vertical group stacks its children vertically.

The `box-direction` CSS property specifies the direction of the group. It has values of `normal`, `reverse`, and `inherit`. A normal horizontal group lays out its children from left to right, and a normal vertical group lays out its children from top to bottom.

Direction within nested groups is inherited by default. If direction is omitted on the outermost group, it is assumed to have a horizontal orientation and normal direction.

```
<group>
  <textbox  ref="/person/name">
    <caption>Please enter your name</caption>
  </textbox>
  <textbox ref="/person/ssid">
    <caption>Enter your SSID</caption>
  </textbox>
</group>
```

When items are placed in a group without specifying any additional information about their size, then the sizes the children *intrinsically*. In other words, the group makes each child only as big as it needs to be. Notice that in the above example, the two form controls are only as big as they need to be, and that this in turn determines the size of the group (since we assume for simplicity that this group is not nested).

Widths can be specified for items inside a horizontal group. When an object specifies its width in CSS, it is telling the group that it would like to be that width. Similarly, heights can be specified in CSS for items in a vertical group. A non-nested group placed inside an enclosing CSS block will obey all the usual sizing rules of the block. For example, setting the width of a non-nested group to 100% ensures that the group is the maximum width permitted by the enclosing CSS block.

8.4.2 Alignment

The `box-align` property specifies how controls are aligned along the orientation of the group. Its possible values are `left`, `center`, `right`, `top`, `bottom`, `baseline`, `stretch` and `inherit`. The default value is `stretch`.

By default a horizontal group ensures that all children are the same height. Once a height is computed for a horizontal group, all of the children are stretched vertically to the height of the group (minus the CSS border and padding on the group). Similarly, a vertical group ensures that all its children are the same width.

The stretch policy overrides any specified width or height values.

```
<group style="box-orient: vertical">
  <output ref="/person/name"/>
  <output ref="/person/age"/>
  <output ref="/person/country"/>
</group>
```

In addition to `stretch`, a horizontal group can also align its children using the values `top`, `center`, `baseline` and `bottom`. A vertical group can align its children using the values `left`, `center`, and `right`. When these values are used, the items are no longer stretched. They are sized intrinsically and then aligned on the group axis according to the specified property value on the containing group.

8.4.3 Controlling Automatic Sizing

The layout algorithm can be controlled by specifying the degree to which items may *flex* i.e., the degree to which an item allows itself to be *auto-sized*. Items in groupes fall into two categories: `flexible` and `inflexible`. Inflexible objects will not grow, even when there is extra space left over in the group. When several objects in a group are flexible, the extra space is divided among the objects based on how flexible they are. The group determines how much space to give an object by adding up the flex values on all of its children. An object gets an amount of extra space equal to the percentage of flex that it is responsible for.

For example, if two objects have a flex of 1, then after both objects are given their preferred sizes, the first object gets 1/(1+1) = 1/2 of the extra space, and the second object also gets 1/2 of the extra space.

```
<group style="box-orient: horizontal; width: 200px">
  <textbox  style="box-flex: 1"  ref="/person/name>
    <caption>Please enter your full name: </caption>
  </textbox>
  <textbox  style="box-flex: 1.5" ref="/person/age>
    <caption>How young?</caption>
  </textbox>
</group>
```

8.4.4 Preferred, Minimum, and Maximum Sizes

For inflexible objects, the specification of the `width` and `height` CSS properties can be used to specify a preferred size. If these properties are omitted, an inflexible object will be sized intrinsically, i.e., it will be given the minimum amount of space required to lay out the item.

With flexible objects, there are more options. Just as with inflexible objects, the `width` and `height` properties can be used to specify a preferred size. Unlike inflexible objects, however, this size is only a guideline. A flexible object will stretch as necessary. It will also shrink if required until it can shrink no more (e.g., when the object hits its minimum required intrinsic size).

```
 <group style="box-orient: vertical">
  <textbox style="box-flex: 1; height:1000px" ref="/person/name">
    <caption>Please enter your name:</caption>
  </textbox>
</group>
```

In the above example if the group were to shrink, the textbox being flexible would shrink--despite its preferred height of `1000px`. The group continues to shrink minimum required height for the textbox is reached. After that, the textbox can shrink no further. If the group were to continue to shrink, the form control's contents would be clipped, and a portion of the control would no longer be visible.

For a more fine-grained control of minimum and maximum sizes, the `min-width`, `min-height`, `max-width`, and `max-height` CSS properties can be used. When specified, these properties provide extra information to the group as it shrinks and grows the object in question.

In a horizontal group, for example, if a minimum width is specified, then the flexible object will not be allowed to shrink below that width. If a maximum width is specified, then the flexible object will not be allowed to grow beyond that width.

The above example demonstrates the use of `min-height` and `max-height` within a vertical group. In the first image the group has been shrunk until it is smaller than 100 pixels in height. Because the `iframe` has a specified minimum height of 100 pixels, it is not allowed to shrink any further, and so as the group falls below 100 pixels, the `iframe` gets clipped, and portions of it become invisible.

In the second image, the group has been expanded past 300 pixels in height. Even though the group is getting bigger, the extra space is not being consumed by the `iframe`, because its maximum height of 300 pixels has been reached. It will grow no further.

Below is another example illustrating min and max width.

```
Example unavailable at publication time
```

In the above example, the group has been stretched so that it is very wide. The first child has a maximum width of 50 pixels, and it divides the excess space equally with the second child until its maximum width has been reached. After that, since it is not allowed to grow any further, the remaining space all goes to the second child.

8.4.5 Packing Controls

When all of the items within a `group` element are inflexible or when all objects have grown to their maximum sizes and can stretch no further, extra space may be left over in the group. An additional property can be used to dictate how any additional space should be distributed between objects. The CSS property `box-`

`pack` has the possible values of `left`, `right`, `top`, `bottom`, `center`, `justify` and `inherit`. The default is `justify`.

In the above example, the button is centered within the group using the `box-align` and `box-pack` properties together. The former centers the button vertically, and the latter centers the button horizontally.

8.4.6 Overflow

Items within a group may use the CSS `overflow` property to obtain horizontal and vertical scrollbars as needed. Flexible objects that shrink below their minimum intrinsic size (but that still have a size greater than a specified CSS minimum) can display scrollbars using the `overflow` property. If overflow is not specified, the object will be clipped instead.

8.4.7 Inlines and Blocks

Whether an element is inline or block when placed directly inside a group is irrelevant. Objects will be flowed horizontally or vertically based off the `box-orient` property.

When any raw text nodes are encountered directly inside a group, an anonymous block is constructed to wrap the text. This anonymous block then participates in the layout as a single item.

8.5 Multiple Sub-forms Or Sub-pages

Editorial note: Raman	March 12, 2001
Notice that we originally started by creating an equivalent of `fieldset`--but given the present design of the XForms UI layer, and given that conditional construct `case` can take either `group` or `html:div` elements, there is little left in this construct that is additional to what is in XHTML `html:div`. I therefore suggest simply reusing `html:div`--perhaps bind XForms specific behaviors through CSS e.g., a stack of cards for example? The way I am thinking of this is that whether it is a stack of cards shown one at a time, or a sequence of tab dialogs is a presentation issue and might therefore be best left to CSS as we have done with the rest of the presentational issues in XForms.	

8.5.1 Subpages

Subpages provide a means to present XForms one bit at a time, breaking a complex task into smaller, simpler parts. Presentation of a subpage can occupy the entire "page" or just part of a page. Different presentations are possible, e.g. a stack of *pages* with visible name tags, or as a set of buttons for flipping through the stack or navigating directly to a particular subpage. One possible representation is a `formset` element enclosing one or more `subpage` elements, each of which starts with a `caption` element.

As the name implies `subpage` is not specific to XForms--our intent is to design `subpage` so that it can be used within XForms--and more generally within XHTML to create presentations where document views are presented to progressively

reveal the document structure and content.

9 Binding

9.1 Introduction

Binding is the glue that connects the separate pieces of XForms--the XForms Model, instance data, and form controls. The binding is independent of the user interface language used.

Binding is specified through the use of binding expressions. The syntax and details of binding expressions are defined in the chapter **6 XPath Expressions in XForms**. This chapter describes the wider topic of how binding expressions are used within XForms.

A future revision of this chapter will address binding across XForms Models, for instance declaring an "address" in one XForms Model and referencing it from another.

9.2 Binding Attributes

XForms defines an attribute `ref` that can be placed on any form control. Note that when placed on form controls outside of XForms, it must be appropriately namespace-qualified. The value of the attribute is a binding expression, based on **6 XPath Expressions in XForms**, that links the form control to a particular location in the instance data (and therefore a particular model item). For example:

Example: XForms User Interface Markup with Binding Attributes

```
<xform:textbox ref="binding-expression">
   <xform:caption>Your first name</xform:caption>
</xform:textbox>
```

The `ref` attribute links the form control to the instance data and XForms Model declared elsewhere in the containing document.

This can also be used on non-XForms form controls, for instance XHTML:

Example: XHTML with Binding Attributes

```
<html:input type="text" name="..." xform:ref="binding-expression"/>
```

Here the `ref` attribute links an XHTML form control to the instance data and XForms Model contained elsewhere in the containing document. Note that the `html:` prefix is used here to represent the XHTML namespace.

Details on the operation of binding expressions are given throughout the rest of this chapter.

9.3 Direct Binding

When a containing document has only a single XForms Model and only a single set of instance data, binding is simpler because there is no possibility of ambiguity as to which XForms Model and instance data will participate. The following syntax can be used for the binding expression:

- A binding expression that evaluates to a node-set in the instance data. If the node-set contains more than one node, only the first node is used. If the node-set contains zero nodes, the binding expression is invalid.

For example:

Example: Binding Expression

```
<xform:textbox ref="orderForm/shipTo/firstName">
...
```

Here the `ref` attribute specifies a path through the instance data to the desired location.

```
<orderForm>
  <shipTo>
    <firstName>value</firstName>
  </shipTo>
</orderForm>
```

Here is the matching instance data for the above example.

A special case applies when binding to an element in the instance data that contains an attribute of type `xsd:ID`. In this case, an XPath function `id()`, can be used:

Example: Binding Expression with XPath id() Syntax

```
<xform:textbox ref="id('myfirstname')">
...
```

Here the `ref` attribute specifies a link to an instance data element with an id of `myfirstname`.

```
<a>
  <b id="myfirstname">value</b>
</a>
```

Here is the instance data for the above example.

For this syntax to be valid, the following conditions must be true:

- The instance data must be included in the same document as the XForms User Interface
- Every referenced element in the instance data must be marked with a valid ID (i.e. the ID is unique throughout the whole document)

Note also that:

- this method is limited, because it requires exactly one instance, decorated with ID attributes.
- It is still legal to have elements without ID attributes in the instance data.
- Only elements can be addressed

9.4 Indirect Binding

In situations where a form is designed by collaboration (such as between a graphic designer and a database/XML specialist) it is desirable to locate all binding expressions in a single area in the containing document. XForms allows the binding expression to appear in a separate element `bind`, a child of the `xform` element.

The attributes of `bind` are `id` of type `xsd:ID` and `ref` which takes a binding expression. When a binding expression is defined this way, the form control can reference the `id` of the `bind` element, as seen here:

Example: Binding Expression Using Indirection

```
<xform:textbox ref="id('myfirstname')">
...
```

Here the `ref` attribute specifies a link to a binding expression defined elsewhere.

```
<xform:bind id="myfirstname" ref="orderForm/shipTo/firstName"/>
<xform:instance>
  <orderForm>
    <shipTo>
      <firstName>value</firstName>
    </shipTo>
  </orderForm>
</xform:instance>
```

Here is the instance data for the above example.

This accomplishes the following:

- It is no longer necessary to add IDs to the instance.
- The binding mechanism is *independent* of the schema and user interface mechanisms.

9.5 Multiple Forms per Page

One design goal of XForms is to support multiple forms per page. This is accomplished by having multiple `xform` elements within a containing document. Each `xform` element might have a separate XForms Model or instance data defined. This makes binding slightly more complex, because the correct `xform` element needs to be referenced.

By default, binding expressions are based on the instance data associated with the first `xform` element in document order. To refer to the contents of subsequent `xform` elements, these must be decorated with `id` attributes of type `xsd:ID`. Attached to the form control element, an additional attribute `xform` serves to set the context node for the attached binding expression to the instance data or XForms Model associated with the `xform` element with the matching `id`.

For example:

Example: Binding Expression Specifying Non-default `xform` Element

```
<xform:textbox xform="b" ref="/orderForm/shipTo/firstName">
...
```

Here the `xform` and `ref` attributes specify a binding to the instance data and XForms Model.

```
<xform:xform id="b">
  <xform:model>
    ...
  </xform:model>
  <xform:instance>
    <orderForm xmlns="...">
      <shipTo>
        <firstName>value</firstName>
      </shipTo>
    </orderForm>
  </xform:instance>
</xform:xform>
```

Scoped resolution of binding expressions, as defined in the chapter **6 XPath Expressions in XForms** can in some cases be used to avoid repetitive uses of the `xform` attribute.

10 Document Structure

XForms are an application of XML [XML 1.0], and have been designed for use

within other XML vocabularies, in particular XHTML [XHTML 1.0]. This chapter discusses some of the high-level features of XForms that allow this specification to be used with other document types.

10.1 The XForms Namespace

The XForms namespace has the URI: `http://www.w3.org/2001/06/xforms`. Future revisions are expected to use a different identifier. This document uses the convention of an `xform:` prefix to represent elements and attributes that are part of the XForms Namespace.

XForms Processors must use the XML namespaces mechanism [XML Names] to recognize elements and attributes from this namespace. Except where specifically allowed by the Schema for XForms, foreign-namespaced elements are not allowed as content of elements in the XForms namespace. Foreign-namespaced attributes are, however, allowed on any XForms element. The XForms Processor must ignore any foreign-namespaced elements or attributes that are unrecognized.

10.2 XForms Elements

10.2.1 xform

The `xform` element is used as a container for other XForms elements, and can serve as the root element of a standalone document or be embedded in other document types such as XHTML. A single containing document may contain any number of `xform` elements.

Editorial note	
under discussion are XLink attributes on the `xform` element. These are: `xlink:type="extended"` and `xlink:role="http://www.w3.org/2001/06/xforms"` - and they should be defaulted or even fixed in the Schema/DTD.	

Example: XML Representation: <xform>

```
<xform
  xmlns = namespace-identifier
  id = xsd:ID
>
  <!-- Content: (submitInfo | bind)*, model, instance -->
</xform>
```

xmlns = namespace-identifier - Optional standard XML attribute for identifying an XML namespace. It is often useful to include this standard attribute at this point.
id = xsd:ID - Optional unique identifier used to refer to this particular `xform` element.

For example:

```
<xform xmlns="http://www.w3.org/2001/06/xforms" id="Person">
   <model xlink:href="Schema-Questionnaire.xform" />
   <instance xlink:href="URL-to-retrieve-defaults" />
   ...
</xform>
```

10.2.2 model

The `model` element is used to define the XForms Model. The content of the XForms Model may be defined inline or obtained from a external URI.

Example: XML Representation: <model>

```
<model
  id = xsd:ID
  xlink:href = xsd:anyURI
>
  <!-- Content: ( schema subset syntax ) -->
</model>
```

id = xsd:ID - Optional unique identifier.
xlink:href = xsd:anyURI - Optional link to an externally defined XForms Model.

Editorial note	
As above, we need to find a place to discuss the defaulted attributes. Here they are `xlink:role="http://www.w3.org/2001/06/xforms-model" xlink:type="locator"`	

10.2.3 instance

The `instance` element is used to define initial instance data. The instance data may be defined inline or obtained from a external URI.

Example: XML Representation: <instance>

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

id = xsd:ID - Optional unique identifier.
xlink:href = xsd:anyURI - Optional link to externally defined instance data

The content of the `instance` element is arbitrary XML in any namespace other

than the XForms namespace. Authors must ensure that proper namespace declarations are used for content within the `instance` element.

Editorial note	
As above, we need to find a place to discuss the defaulted attributes. Here they are `xlink:role="http://www.w3.org/2001/06/xforms-instance" xlink:type="locator"`	

Issue (issue-schemalocation):

Should a `schemaLocation` attribute, linking the instance data with a schema definition, be present here?

10.2.4 submitInfo

The `submitInfo` element provides information on how and where to submit the instance data.

Example: XML Representation: <submitInfo>

```
<submitInfo
  id = xsd:ID
  xlink:href = xsd:anyURI
  method = xsd:string
>
  <!-- Content: (##empty) -->
</submitInfo>
```

id = xsd:ID - Optional unique identifier.
xlink:href = xsd:anyURI - Required destination for submitted instance data.
method = xsd:string - Optional indicator to provide details on the submit protocol. With HTTP, the default is "`POST`".

Issue (submit-method-values):

The possible values for `method`, and their respective meanings, still need to be defined.

10.2.5 bind

The `bind` element represents a connection between the different parts of XForms.

Example: XML Representation: <bind>

```
<bind
  id = xsd:ID
  ref = XForms binding expression
>
  <!-- Content: (##empty) -->
</bind>
```

id = xsd:ID - Required unique identifier.
ref = XForms binding expression - A link to an externally defined XForms Model.

Additional details are found in the chapter **9 Binding**.

10.3 Integration with XLink

XForms make good use of XLink [XLink] features. To that end, the XLink namespace is integrated in the DTD/Schema, the majority of the attributes have sensible defaults, including those based on XLink roles defined below.

This document uses the convention of an `xlink:` prefix for the XLink namespace (informationally: as of this writing, the XLink namespace identifier is `http://www.w3.org/1999/xlink` - this may change when XLink reaches W3C Recommendation state. Please verify at [XLink])

Note that the XLink support uses a well-defined XLink failure mode: If an XLink attribute is not provided, the element looses its XLink specific meaning. We use this feature in order to allow application developers to either provide the model and instance via an external reference (via an `xlink:href` attribute) or to provide the data inline without the attribute. In the latter case, the XLink-specific meaning of the element is lost and the inline content used. If both inline content and external reference is provided, a processor must use the external reference and ignore the inline content.

For the purposes of XForms, we suggest that XLink aware processors switch from the `xlink:type="locator"` mode to the `xlink:type="resource"` mode. This should be specified in the document by setting `xlink:type="resource"`, though a processing agent may not depend on it. In other words, the first two of the following examples must be treated identically:

Example: Inline XForms Model, without explicit change to the `xlink:type`

```
<model>
  <!-- Content: ( schema subset syntax ) -->
</model>
```

Example: Inline XForms Model, with explicit change to the `xlink:type`

```
<model xlink:type="resource">
  <!-- Content: ( schema subset syntax ) -->
</model>
```

Example: External XForms Model

```
<model xlink:href="URI" />
```

XLink as originally specified allows users to provide arc-type elements to specify traversal rules. The integration of arc-type elements in XForms would require additional elements in the `xform` element that are otherwise not necessary for XForms. Hence, for children of the `xform` element, the traversal rule is to traverse `xlink:from` the current document `xlink:to` the document pointed to by the external resource. The processor should behave as if `xlink:actuate="onLoad"` was specified. The `xlink:show` attribute is meaningless in this context, anyway.

An XForms processor is not required to implement full XLink--correct behavior of the `xlink:href` attribute (as defined above) is sufficient. It is permissible to construct the additional information from the semantics of the elements. An XForms Processor can not be XForms compliant, however, if it attempts to implement XLink and the implementation does not conform to XLink specification with respect to the attributes used by XForms.

The following definition of the XLink roles defines the relationship between the various resources participating in a XForms-based form, not properties that are inherent to the resources. It is perfectly permissible for the same resource to participate in various XForms-based forms in different roles.

10.3.1 XLink role for XForms

The xlink-role for XForms Models is `http://www.w3.org/2001/06/xforms`. This means that XLink processors encountering a link with a `xlink:role="http://www.w3.org/2001/06/xforms"` must assume that the arcs associated with the other XLink roles for XForms (see below) behave as above. In other words, this XLink role is an additional hint beyond the element names for an XLink processor to identify the arcs. For all roles, if element names and XLink behavior conflict, XLink role behavior prevails.

10.3.2 XLink role for the XForms Model

The xlink-role for XForms Models is `http://www.w3.org/2001/06/xforms-model`. This means that XLink processors encountering a link with a `xlink:role="http://www.w3.org/2001/06/xforms-model"` must assume that the referenced resource relates to the other resources as XForms Model as defined in **5 The XForms Model**.

10.3.3 XLink role for the Instance Data

The xlink-role for XForms Instances is `http://www.w3.org/2001/06/xforms-instance`. This means that XLink processors encountering a link with a `xlink:role="http://www.w3.org/2001/06/xforms-instance"` must assume that the referenced resource is the initialization data of a form.

10.3.4 XLink role for the XForms User Interface

The xlink-role for XForms User Interfaces is `http://www.w3.org/2001/06/xforms-ui`. This means that XLink processors encountering a link with a `xlink:role="http://www.w3.org/2001/06/xforms-ui"` must assume that the referenced resource is the user interface of a form. Note that this role does not make an assertion about the content type of the referenced resource other than that it is an XML format. For example, it can be XHTML+XForms, pure XForms, SVG+XForms or pure SVG any other combination. This role does not correspond to any element defined in XForms; it is defined so that XLink linkbases [XLink] can be established containing all the information about a XForms document.

11 Processing Model

11.1 Introduction

The XForms Reference Processing Model is a normative explanation of the components, predictive behavior, and mechanisms of XForms Processors. It is not intended to constrain implementations. XForms Processors may be implemented in any manner, so long as the end results are identical to that described in this chapter.

This chapter uses the terms **may**, **must**, and **should** (when rendered as in this paragraph) in accord with RFC 2119.

> **Issue (issue-processing):**
>
> This chapter is still at an early phase and may contain errors or omissions. Feedback on this chapter is especially appreciated.

11.1.1 Design Rationale

The Reference Processing Model set out in this chapter will:

- Be simple enough to implement across a wide range of devices, including resource-constrained handhelds and appliances.
- Define a predictive processing model with enough detail for implementors to create interoperable software.
- Define a well-ordered system for calculations and dependencies independent of processor speed or threading.
- Provide a unified addressing scheme for binding expressions, independent of how the structure of the instance data is defined.
- Be simple enough for the existing base of HTML authors to quickly get up to speed.

- Be compatible (to the extent reasonably possible) with existing form processing.

11.2 XForms Properties

For each `xform` element, the XForms Processor maintains a set of read-write properties, as described here. These properties are available to all expressions in the containing document.

- `immediate-refresh`
- `immediate-revalidate`
- `immediate-recalculate`
- `use-nil`

`immediate-refresh` controls whether changes in the instance data are immediately updated in the UI

`immediate-revalidate` controls whether changes in the instance data immediately trigger a validation

`immedate-recalculate` controls whether changes in the instance data immediately trigger a recalculation

`use-nil` controls whether XML Schema Instance nils are placed in the instance data

Additionally, the following properties are available for reading (but not modification). These properties are available to all expressions in the containing document.

- `version`
- `conformance-level`
- `timezone`

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

`conformance-level` strings are defined later in this chapter

`timezone` strings are signed integers representing the number of minutes offset from GMT

11.3 Events

XForms uses an events system as defined in [DOM2 Events], with a Capture phase, arrival at the Event Target, and then a Bubbling Phase.

Events fall into different groupings. One class of events indicates that some processing is about to happen. That processing may be halted by the event handler:

- `xforms-submit`
- `xforms-reset`
- `xforms-value-changing`
- `xforms-interactive-value-changing`
- `xforms-instance-changed`

Another class of events indicates that some processing has already happened or is in progress. Such processing can not be halted by the event hander:

- `xforms-construct`
- `xforms-destruct`
- `xforms-initialize`
- `xforms-exception`

Finally, certain events are used by the author or the XForms Processor to cause processing to happen:

- `xforms-recalculate`
- `xforms-refresh`

Unless otherwise noted, the target node for all events is the `xform` element. When a containing document has multiple `xform` elements, the binding is used to determine which `xform` element is used.

The Working Group is using pre-defined generic event handling, defined in [XHTML Events], additionally defining a set of XForms-specific actions.

11.4 XForms Processing

11.4.1 Initialization/Resume

The following describes the initialization process for XForms. Initialization must occur before any other processing. For each `xform` element in the containing document, in document order, the following processing occurs:

1. An `xforms-construct` event is fired; this is the place for authors to handle any initialization tasks.
2. Instance data is constructed (**11.4.2 Instance Data Construction**).
3. An `xforms-initialize` event is fired. A handler for this event could perform form initialization tasks such as a database lookup.
4. A recalculation (**11.4.5 Recalculation Algorithm**) takes place.
5. A UI refresh (**11.4.6 UI Refresh Algorithm**) takes place.

11.4.2 Instance Data Construction

The following steps describe how the instance data associated with each `xform` element is constructed. Of the following options, the first applicable option is

chosen. Only one of the following applies:

1. If an `instance` element is present and contains non-whitespace child nodes, the contents of the `instance` element are copied into the instance data tree, based on the infoset mappings defined in the XPath [XPath 1.0]data model.
2. If an `instance` element is present and contains a reference to non-local initial instance data, it is retrieved by traversing the link to it, then copied into the instance data as described above. A remote instance that is unretrievable for any reason is ignored, in which case an XForms Processor **may** issue a warning.
3. If an `instance` element is not present, then a default instance data configuration is produced, according to the following rules:
    1. Each form control bound to the `xform` element currently being processed is visited in document order. Each form control's binding expression is evaluated.
    2. If the instance data item result of evaluating the binding expression doesn't already exist, it is created, and if the `use-nil` property is true, populated with a nil value (an `xsi:nil="true"` attribute). Note that only elements can hold nil values. The form control receives a default blank value. The algorithm for creating instance data items is as follows: For each location step in the canonical binding expression, from left to right, where no matching node exists in the instance data, a new node is inserted.

        Issue (creating-instance-nodes):

        The algorithm for creating instance nodes is under discussion, with one possibility being ignoring the path information, using only the local name, in a flat list.
4. If none of the above options are fulfilled, this is an error condition, and the XForms Processor **must** stop processing with an error message.

11.4.3 Navigation Sequence Algorithm

Navigation is determined on a containing document-wide basis. The navigation sequence is determined as follows:

1. Those form controls that support `navindex` and assign a positive value to it are navigated first. Navigation proceeds from the form control with the lowest `navindex` value to the form control with the highest value. Values need not be sequential nor must they begin with any particular value. form controls that have identical `navindex` values should be navigated in document order.
2. Those form controls that do not supply `navindex` or supply a value of "0" are navigated next. These form controls are navigated in document order.
3. Those form controls that are disabled, hidden, or on a non `relevant` subtree are assigned a relative order in the overall sequence but do not participate as navigable controls.
4. The navigation sequence past the 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.

11.4.4 Interactivity

XForms provides similar processing to the HTML `onChange` event. As users indicate completion of a form control by navigating away the following occurs:

1. If the display value has changed since the user last navigated to the form control, an `xforms-value-changing` event is fired. If the display value hasn't changed, processing for this event ends.
    1. Any listener may prevent default processing (one option under consideration provides a `<stopevent/>` action), which will end event processing immediately after the Capture and Bubbling phases. Alternatively, a listener may perform a custom translation from display value to canonical value. Any listener may have side-effects that modify any instance data item, in which case the modified instance data items must be marked "dirty".
    2. Default processing is to convert the display value of the form control to the canonical value as specified in the Datatypes chapter. Default processing should automatically take into account regional settings (if any), such as decimal character symbol, date formats, etc.
2. If the `immediate-revalidate` property is true, all validations (**11.4.7 Revalidation Algorithm**) bound to the form control are run. Note that validation is performed against the canonical value, not the display value.
    1. If any validation fails, the user **must** be notified, and **may** not be allowed to navigate away from the control. The invalid entry in the form control **should** be preserved. The associated instance data item is left unchanged, thereby ending processing for this event.
3. The instance data item is updated with the new value, and marked "dirty".
4. If the `immediate-recalculate` property is true, a recalculate (**11.4.5 Recalculation Algorithm**) occurs to perform any defined calculations.
5. If the `immediate-refresh` property is true, a refresh (**11.4.6 UI Refresh Algorithm**) occurs to update any form controls that might be dependent on this newly changed value.

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

Example: A partially entered currency value of "U" is not valid because it doesn't (yet) have 3 characters. This is permitted temporarily, as long as the user remains on the form control. XForms Processors with sufficient processing resources would typically update/refresh on every character. Resource-constrained XForms Processors would typically only update/refresh on the final value.

1. Any time the display value of a form control changes (such as through character or cut/paste activities), even without indication that this is a final value, an `xforms-interactive-value-changing` event is fired. Resource-constrained XForms Processor implementations **may** choose to ignore all such events.

1. Event listeners may prevent default processing.
2. Otherwise, default handling is as follows: The current form control is revalidated (**11.4.7 Revalidation Algorithm**). This is for internal purposes only, and happens regardless of the `immediate-revalidate` setting. If all validations on the form control are successful, the instance data item is updated, and marked "dirty". If any validations fail (indicating a transitional value) all form controls bound to the same instance data item **may** be directly updated with the display value. Otherwise, the following occurs:
3. If the `immediate-recalculate` property is true, a recalculation (**11.4.5 Recalculation Algorithm**) occurs to perform any defined calculations.
4. If the `immediate-refresh` property is true, a refresh (**11.4.6 UI Refresh Algorithm**) occurs to update any form controls that might be dependent on this newly changed value.

Implementations that choose to respond `xforms-interactive-value-changing` are expected optimize processing (for instance not flashing the entire screen for each character entered, etc.).

11.4.5 Recalculation Algorithm

XForms Processors are free (and encouraged) to skip or change any steps in this algorithm, as long as the end result is the same. Each form control may have a model item property `priority` value, which is the main factor in determining calculation order.

Following is the default handling for an `xforms-recalculate` event:

1. Each model item with a bound `calculate` model item property is visited in **calculation order**, which is defined as follows:
    1. Those model items that are bound to a `priority` and assign a positive integer to it are computed first. Computation proceeds from the model item with the lowest bound `priority` to the model item with the highest bound `priority`. Values need not be sequential nor must they begin with any particular value. Model items with the same bound `priority` value are computed in document order.
    2. Those model items not bound to a priority or bound to one with the value "0" are computed next. These model items are computed in document order.
2. For each model item, the expression in the `calculate` model item property is evaluated. Any instance data item changes as a result of this are marked with a "dirty" flag.
3. The instance data item bound to the model item is updated with the result of the `calculate` expression, and the "dirty" flag is set.

11.4.6 UI Refresh Algorithm

Following is the default handling for an `xforms-refresh` event:

1. For purposes of UI refresh, the instance data as it exists at the beginning of

processing the `xforms-refresh` event is used.

2. Each form control is visited in refresh order, which is defined as follows:
    1. Those form controls that have a given or computed navigation sequence value are visited first, in the navigation sequence.
    2. Those form controls outside the navigation sequence are visited next. These form controls are visited in document order.
3. For each form control, the `relevant` constraint is evaluated, which might result in the form control being disabled/hidden/etc. as specified in the chapter **5 The XForms Model**.
4. For each form control, the binding expression is evaluated. If the instance data indicates that the instance data item is not "dirty", processing for that particular form control completes.
    1. Otherwise, if the instance data item is "dirty", an `xforms-instance-changed` event is fired.
    2. Listeners to the `xforms-instance-changed` event are free to compute a new display value.
    3. Listeners to the `xforms-instance-changed` event are prohibited from directly updating any form controls present.
    4. Listeners to the `xforms-instance-changed` event are prohibited from altering any portion of the instance data. To attempt to do so results in an `xforms-exception` being fired.
    5. Listeners may prevent the default processing of the `xforms-instance-changed` event.
    6. Default processing is to convert the canonical value into a display value, taking into account regional settings (if any) such as decimal separator character, etc.
5. The form control is updated with the display value.
6. After all form controls have been updated, all "dirty" flags in the instance data are cleared.

Note:

Editor’s Note: Still to be addressed is the processing when a datatype facet or model item property are changed--what gets marked "dirty"?; what gets recalculated?; what gets revalidated?; what gets refreshed?

11.4.7 Revalidation Algorithm

Revalildation always occurs within the scope of a context form control. Following is the revalidation process:

1. The bound instance data item is checked against any bound XForms Datatype constraining facets. If any fail, the context form control is considered invalid.
2. The bound instance data item is checked against any bound Schema Datatype constraining facets. If any fail, the context form control is considered invalid.
3. If a `validate` model item property is bound to the context form control, the

expression within is evaluated. If it evaluates to false, the context form control is considered invalid.

4. If the context form control is invalid, the XForms Processor **must** notify the user. The XForms Processor **may** combine messages before presentation to the user.

11.5 Submit and Reset

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

11.5.1 Submit

In response to an `xforms-submit` event, the following takes place:

1. Event listeners may prevent default processing of the submit request. Otherwise, default handling as described below occurs.
2. Every form control is revalidated (**11.4.7 Revalidation Algorithm**). Any invalid values **must** be reported to the user and submit processing **must** not continue.
3. A subset or all of the instance data is selected based on the binding expression used to invoke the submit request. The selected nodes and all children are selected for serialization as submitted data. If no `ref` attribute is specified, all nodes in the instance data are selected by default.
    1. If the instance data selection results in an empty node-set, the submit **must** be aborted and submit processing **must** not continue.
4. Instance data is serialized according to one of the processes defined below.
5. Instance data is delivered over the network as an HTTP POST.
6. Upon successful delivery of the submit data, an `xforms-destruct` event is fired and form processing shuts down.
7. The response page sent by the server replaces the current containing document.

Issue (method-strings):

We have yet to define the method strings (e.g. `method="post"` in XHTML)

11.5.2 Reset

In response to an `xforms-reset` event, the following takes place:

1. Event listeners may prevent default processing of the reset request. Otherwise, default handling as described below occurs.
2. A subset or all of the instance data is selected based on the binding expression used to invoke the suspend request. The selected nodes and all children are selected for resetting. If no `ref` attribute is specified, all nodes in the instance data are selected by default.
    1. If the instance data selection results in an empty node-set, the reset

has no effect.

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

11.6 Serialization Formats for Instance Data

11.6.1 application/x-www-form-urlencoded

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

This format is not suitable for the persistence of binary content. Therefore, it is recommended that XForms capable of containing binary content use either the multipart/form-data (**11.6.2 multipart/form-data**) or text/xml (**11.6.3 text/xml**) formats.

Issue (issue-urlencoding-mods):

Modifications to urlencoding process

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

Issue (issue-utf8-encoding):

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

The steps for building this persistence format is as follows:

1. Prepare a new UTF-8 encoded string buffer to hold the persisted instance data.
2. Beginning with the root element of the instance data, iterate over the selected content of the instance data in document order and build an ordered set of strings by performing the following steps:
    1. For each element with an attribute, append to the set a string of the format "*path*=*value*" where *path* is the canonical binding expression that refers to each attribute, and *value* is the character content of each attribute (urlencoded if necessary).
    2. For each element enclosing character content, append to the set a string of the format "*path*=*value*" where *path* is the canonical binding expression that refers to the element, and *value* is the character content of the element (urlencoded if necessary).

3. For each element enclosing element content, continue the iteration.

3. Append the strings from the ordered set together, delimiting the strings with an ampersand '&' character, and place the result of the append into the UTF-8 encoded string buffer.

Example:

Example: application/x-www-form-urlencoded

```
/PersonName/@title=Mr&/PersonName/FirstName=Roland
```

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

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

Here is the instance data for the above example.

11.6.2 multipart/form-data

This format is intended to facilitate the integration of XForms into HTML forms processing environments, and represents an extension of the [XHTML 1.0] form content type of the same name that expresses the hierarchical nature of instance data. Unlike the application/x-www-form-urlencoded (**11.6.1 application/x-www-form-urlencoded**) format, this format is suitable for the persistence of binary content.

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

Example:

Example: multipart/form-data

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

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

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

Roland
--AaB03x

...Possibly more data...

--AaB03x-
```

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

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

Here is the instance data for the above example.

11.6.3 text/xml

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

The steps for building this persistence format is as follows:

1. Prepare a new empty XML document to hold the persisted instance data.
2. If the selected content of the instance data corresponds to a singly-rooted data structure, serialize, into the XML document the entire content of the selected instance data, beginning at the root node.
3. If the selected content of the instance data corresponds to a multiply-rooted data structure (such as a general parsed entity), an unqualified root element of <`Envelope`>, with an unqualified element <`Body`> is inserted into the XML document, and the selected instance data serialized into the content of the <`Body`> element.

11.6.3.1 Binary Content

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

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

11.7 Conformance

XForms are being designed for use on hardware platforms of all sizes, from tiny handheld devices to high-powered servers. Clearly, a one-size-fits-all approach has its drawbacks. For this reason, the XForms Working Group has begun specifying two conformance levels for XForms Processors, documents, and authoring tools.

11.7.1 XForms Basic

This conformance level will be suitable for devices with limited computing power, such as mobile phones, handheld computers, and appliances. This conformance level will depend on a subset of XML Schema, and will not include any resource-intensive features. Implementations of XForms Basic should return "`basic`" for the `conformance-level` property.

11.7.2 XForms Full

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

Additional details will be provided in future revisions of this chapter.

A Schema for XForms

```
<?xml version="1.0" encoding="utf-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Micah (W3C
<!-- converted with http://www.w3.org/2001/03/webdata/xsu -->
<!DOCTYPE xsd:schema
  PUBLIC "-//W3C//DTD XMLSchema 200102//EN" "http://www.w3.org/2001/XML
<!ENTITY % p 'xsd:'>
<!ENTITY % s ':xsd'>
]>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xform="http://www.w3.org/2001/06/xforms"
targetNamespace="http://www.w3.org/2001/06/xforms"
elementFormDefault="qualified">
  <!--
structural elements
-->
  <xsd:element name="xform">
    <xsd:annotation>
      <xsd:documentation>Definition of the xform container element.</xs
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xform:submitInfo" minOccurs="0"/>
        <xsd:element ref="xform:model" minOccurs="0"/>
        <xsd:element ref="xform:instance" minOccurs="0"/>
        <xsd:element ref="xform:bind" minOccurs="0" maxOccurs="unbounde
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xform:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="model">
    <xsd:annotation>
      <xsd:documentation>Definition of model container. Content model T
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xform:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
```

```
  <xsd:element name="instance">
    <xsd:annotation>
      <xsd:documentation>Definition of instance container.</xsd:documen
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##any" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attributeGroup ref="xform:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="submitInfo">
    <xsd:annotation>
      <xsd:documentation>Definition of submit info container.</xsd:docu
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##any"/>
        <!-- allow zero or more submitExtension elements -->
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="target" type="xsd:anyURI" use="required"/>
      <xsd:attribute name="method" type="xform:methodType" use="optiona
    </xsd:complexType>
  </xsd:element>
  <xsd:attributeGroup name="linkingAttributes">
    <xsd:attribute name="href" type="xsd:anyURI"/>
  </xsd:attributeGroup>
  <xsd:element name="bind">
    <xsd:annotation>
      <xsd:documentation>Definition of bind container.</xsd:documentati
    </xsd:annotation>
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute name="ref" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>
  <!--
User Interface form controls and other elements
-->
  <xsd:attributeGroup name="commonUIAttributes">
    <xsd:attribute name="lang" type="xsd:language" use="optional"/>
    <xsd:attribute name="id" type="xsd:ID" use="optional"/>
    <xsd:attribute name="class" type="xsd:string" use="optional"/>
    <xsd:attribute name="style" type="xsd:string" use="optional"/>
    <xsd:attribute name="ref" type="xsd:string" use="optional"/>
    <xsd:attribute name="xform" type="xsd:IDREF" use="optional"/>
    <xsd:attribute name="accessKey" type="xsd:string" use="optional"/>
    <xsd:attribute name="navIndex" type="xsd:nonNegativeInteger" use="o
  </xsd:attributeGroup>
  <xsd:element name="caption">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      <xsd:attributeGroup ref="xform:linkingAttributes"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="hint">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      <xsd:attributeGroup ref="xform:linkingAttributes"/>
    </xsd:complexType>
```

```
    </xsd:element>
    <xsd:element name="help">
      <xsd:complexType mixed="true">
        <xsd:sequence>
          <xsd:any namespace="##any"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
        <xsd:attributeGroup ref="xform:linkingAttributes"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="onevent">
      <xsd:annotation>
        <xsd:documentation>Defined in a different specification</xsd:docu
      </xsd:annotation>
    </xsd:element>
    <xsd:group name="choiceGroup">
      <xsd:choice>
        <xsd:element ref="xform:choices"/>
        <xsd:element ref="xform:item"/>
      </xsd:choice>
    </xsd:group>
    <xsd:element name="choices">
      <xsd:complexType>
        <xsd:sequence maxOccurs="unbounded">
          <xsd:group ref="xform:choiceGroup"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="item" type="xform:string"/>
    <xsd:group name="commonUIChildren">
      <xsd:sequence>
        <xsd:element ref="xform:caption"/>
        <xsd:element ref="xform:help" minOccurs="0"/>
        <xsd:element ref="xform:hint" minOccurs="0"/>
        <xsd:element ref="xform:onevent" minOccurs="0"/>
        <xsd:any namespace="##any"/>
      </xsd:sequence>
    </xsd:group>
    <xsd:element name="textbox">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="secret">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="uploadMedia">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
        <xsd:attribute name="mediaType" type="xform:spaceSeparatedListTyp
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="selectOne">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:group ref="xform:commonUIChildren"/>
          <xsd:element ref="xform:choices"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
        <xsd:attribute name="selectUI" type="xform:selectUIType" use="opt
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="selectMany">
```

```
      <xsd:complexType>
        <xsd:sequence>
          <xsd:group ref="xform:commonUIChildren"/>
          <xsd:element ref="xform:choices"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
        <xsd:attribute name="selectUI" type="xform:selectUIType" use="opt
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="selectBoolean">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:group ref="xform:commonUIChildren"/>
          <xsd:element ref="xform:choices"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
        <xsd:attribute name="selectUI" type="xform:selectUIType" use="opt
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="range">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
        <xsd:attribute name="start" type="xsd:string" use="required"/>
        <xsd:attribute name="end" type="xsd:string" use="required"/>
        <xsd:attribute name="stepSize" type="xsd:string" use="optional"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="button">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="output">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attribute name="id" type="xsd:ID" use="optional"/>
        <xsd:attribute name="ref" type="xsd:string" use="optional"/>
        <xsd:attribute name="xform" type="xsd:string" use="optional"/>
        <xsd:attribute name="format" type="xsd:string" use="optional"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="submit">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="reset">
      <xsd:complexType>
        <xsd:group ref="xform:commonUIChildren"/>
        <xsd:attributeGroup ref="xform:commonUIAttributes"/>
      </xsd:complexType>
    </xsd:element>
    <!--
  Bring over xsd simpleTypes
  -->
    <xsd:simpleType name="decimal">
      <xsd:restriction base="xsd:decimal"/>
    </xsd:simpleType>
    <xsd:simpleType name="string">
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="boolean">
      <xsd:restriction base="xsd:boolean"/>
    </xsd:simpleType>
    <xsd:simpleType name="date">
```

```
      <xsd:restriction base="xsd:date"/>
    </xsd:simpleType>
    <xsd:simpleType name="time">
      <xsd:restriction base="xsd:time"/>
    </xsd:simpleType>
    <!-- Note: similar for all other built-in Schema types; new schema wa

  time -->
    <!--
  New simpleTypes
  -->
    <xsd:simpleType name="currency">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[A-Z]{3}"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="maskType">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="((A|a|X|x|N|n|M|m)|(\\.))*([0-9\*](A|a|X|x|N|
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="spaceSeparatedListType">
      <xsd:list itemType="xsd:token"/>
    </xsd:simpleType>
    <xsd:simpleType name="selectUIType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="radioGroup"/>
        <xsd:enumeration value="checkboxGroup"/>
        <xsd:enumeration value="pulldown"/>
        <xsd:enumeration value="listbox"/>
        <xsd:enumeration value="comboGroup"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="methodType">
      <xsd:restriction base="xform:string">
        <xsd:enumeration value="postXML"/>
        <xsd:enumeration value="post"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
```

B References

B.1 Normative References

CSS2

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

DOM2 Events

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

ISO 4217

ISO Standards for Currency Names, International Organization for Standardization (ISO), 1999.

RFC 2388

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

WML1.3
Wireless Application Protocol Wireless Markup Language Specification Version 1.3, Wireless Application Protocol Forum, Ltd., 2000. Available at: http://www1.wapforum.org/tech/documents/WAP-191-WML-20000219-a.pdf.

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

XHTML Events
XHTML Events - An updated events syntax for XHTML, Ted Wugofski, 2000. W3C Working Draft available at: http://www.w3.org/TR/xhtml-events.

XLink
XML Linking Language (XLink) Version 1.0, Steve DeRose, Eve Maler, David Orchard, 2000. W3C Proposed Recommendation available at: http://www.w3.org/TR/xlink/.

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

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

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

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

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

B.2 Informative References

ANSI X3-274
Information Technology - Programming Language REXX., American National Standards Institute (ANSI), 1996. Document Number: ANSI X3.274-1996.

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

ECMA 262
ECMA-262: ECMAScript Language Specification, European Computer Manufacturers' Association (ECMA), 1999. Available at ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf.

Unicode
The Unicode Standard, Version 3.0, Joan Aliprand, Julie Allen, Joe Becker, Mark Davis, Michael Everson, Asmus Freytag, John Jenkins, Mike Ksar, Rick McGowan, Lisa Moore, Michel Suignard, Ken Whistler. Addison-Wesley Developers Press, , Reading, Mass., 2000.

XHTML 1.0

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

XML Schema part 0

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

XSLT

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

C Changes from Previous Release (Non-Normative)

C.1 Changes since the 16-Feb-2001 release

There have been significant changes to the XForms since our last public draft. In addition, we have switched over to authoring the specification using xmlspec.dtd from writing raw HTML (details at **E Production Notes**). The change from hand-authored HTML to xmlspec has introduced changes throughout the document, and consequently, adding diff marks for each change would be excessively distracting and make this appendix difficult to read. Instead, this section summarizes how things have changed so that readers familiar with the previous draft can get up to to speed with the new specification.

C.2 Changes to Chapter 1 ’About XForms’

The typographic samples have been updated to work better with the XMLspec publishing system.

C.3 Changes to Chapter 2 ’Concepts’

This chapter has been completely updated. The new version contains a samples of complete forms and a step-by-step guide through the most common functionality in XForms.

C.4 Changes to Chapter 3 ’Terminology’

Terms have been cleaned-up and brought into line with the rest of the document. The term "datatype" has been imported from XML Schema. The terms "computed expression", "containing document", and "model item property" are new.

C.5 Changes to Chapter 4 ’Datatypes’

The Working Group has decided to more directly reference XML Schema datatypes, hence this chapter has been restructured to include by reference Schema datatypes, and define any additional types. Remaining issues are listed within the chapter.

C.6 Changes to Chapter 5 ’XForms Model’

Removed all "simple syntax". Clarified how computed model item properties (including calculate) work. Improved terminology, consistently using terms from

chapter 3.

Open and closed enumerations are now under separate headings. Removed ’group’ to UI chapter. Renamed "Arrays" section to "Multiple Simultaneous Values", and added section "Repeating Line Items". Renamed "Switch" to "Alternate Representation", with a pointer to the UI section.

Significant changes to this chapter, bringing it into much closer alignment with XML Schema, were not available for publication at this time and will appear in a near-future Working Draft.

C.7 Changes to Chapter 6 ’XPath Expressions in XForms’

This chapter was previously named "Dynamic Constraint Language". The new title more accurately describes both the chapter and the role of XPath within XForms. The term "Dynamic Constraints Language" has been removed throughout the remainder of the document as well.

New introductory material describes the various ways XPath is used throughout XForms. The "Instance Data" section formerly residing in the Processing Model chapter is now in this chapter. The section on Datatypes now describes which parts of XForms use XPath datatypes and which use XML Schema datatypes.

The BNF productions have been removed, in favor of stronger wording that XPath is to be used directly.

The section "Evaluation Context" has been more thoroughly cross-checked against the XPath specification, with several underspecified areas now addressed.

The large set of custom operators has been replaced with a NOTE referencing the built-in XPath operators.

New functions defined--`count-non-empty(node-set)`, and `xforms-property (string)`.

Several of the functions have been specified in greater detail.

The extensibility section now makes a reference to a possible direction--the `element-available()` and `function-available()` functions.

C.8 Changes to Chapter 7 ’Form Controls’

Originally part of the chapter "XForms User Interface". Now split into a separate chapter. The material in this new chapter has been restructured for readability.

The "Design Input" section has been removed. Further details are available in the XForms Requirements document.

Note that names of form controls have changed, but are not yet cast in stone. The following changes and additions have been made to the core form controls:

- hidden has been removed as a form control

output, textbox, submit, and reset have kept their same names
exclusiveSelect and multipleSelect have been renamed selectOne and selectMany
checkbox is now selectBoolean
password is now secret
button has been fleshed out, including event listener details
uploadMedia and range are new form controls in this Working Draft

The "AnyControl" is now described in prose instead of formally. In addition, there are minimal changes to the common elements and attributes defined in this section. Element onevent is now taken from the XHTML Events module and is therefore no longer defined in detail in this specification. A later section defines XForms-specific event handlers for use within event listeners declared via element onevent.

C.9 Changes to Chapter 8 ’XForms User Interface’

The section "Future Work" has been removed.

The specification now introduces constructs for authoring conditional user interfaces (see switch) and a construct for authoring repeating widgets (see repeat) for use in creating purchase orders or shopping carts that wish to create the user interface needed for populating repeating rows from a table.

Conditional User Interfaces: This is a new section, and defines construct switch along with its associated event handlers.

New Construct repeat: There is a new section that defines construct repeat for authoring things like shopping carts where the user interface for populating a structure needs to be repeated multiple times.

UI Templates: Constructs for creating and using user interface templates have been renamed for consistency. The new element names are uiTemplate and useUITemplate.

Constructing High-level User Interfaces: The content pertaining to creating higher-level user interfaces from the basic building blocks has been substantially reorganized. This portion of the specification first defines construct group used to group user interface controls. Element group no longer defines an XForms-specific layout model; instead all layout attributes come from CSS. This enables XForms user interfaces to be laid out using either CSS or XSL:FO, or for that matter languages such as SVG.

Subforms And Subpages: With the XForms constructs for authoring higher level user interaction now fleshed out, it turns out that there is no need for an XForms-specific subform or subpage construct --we can leverage existing mechanisms such as group and more generally, XHTML’s div elements for this purpose.

C.10 Changes to Chapter 9 ’Binding’

Some of the text has been cleaned up with regard to the relationship with XPath, but otherwise no significant content changes.

C.11 Changes to Chapter 10 ’Document Structure’

This chapter was previously named "Using XForms with Other Document Types". Added new section "Integration with XLink". Updated examples to reflect xlink namespace.

C.12 Changes to Chapter 11 ’Processing Model and Conformance’

Cleaned up terminology and brought into greater consistency with the rest of the document.

Removed the `locale` property; separated the rest into those that can vary from one `xform` element to the next, and those that are the same across the containing document. Renamed `use-nulls` to `use-nil`.

The "suspend" functionality has been removed, including the `xforms-suspend` event, and will appear in a post-1.0 version of XForms. The `xforms-resume` event has been renamed `xforms-initialize`. Clarified that model items are associated with only a single value.

The "instance data" section has moved to the XPath Expressions chapter, though rules for constructing instance data are still in this chapter, and reworded for clarity.

The multipart/form-data section has been replaced with a reference to RFC 2388.

An early conformance section has been added.

C.13 Changes to Appendix ’Schema for XForms’

For this release, the Schema only covers the XForms form controls and the Document Structure markup.

Significant changes to this appendix, bringing it into much closer alignment with XML Schema, were not available for publication at this time and will appear in a near-future Working Draft.

C.14 Changes to Appendix ’XSLT from Simple Syntax’

Following the Working Group decision to remove the simple syntax, this appendix has also been removed.

C.15 Change to Appendix ’Sample Forms’

Sample forms have been incorporated into **2 Concepts**

C.16 Changes to Appendix ’Optional Function Library’

The Working Group has decided to remove all optional functions from XForms.

C.17 Changes to Appendix 'References'

References have been updated to refer to the changes in document status, most notably the XML Schema recommendation. New normative reference to XLink, RFC2388; informative reference to T. V. Raman's book on auditory user interfaces. Removed orphaned references to ISO 8601, RFC 2369, RFC 2141 and moved the orphaned Unicode reference to the informative section.

D Acknowledgements (Non-Normative)

The Authors of this document are:

- Steven Pemberton , CWI (*co-chair*)
- Sebastian Schnitzenbaumer , Mozquito Technologies (*co-chair*)
- Micah Dubinko , Cardiff
- Peter Stark , Ericsson
- Roland Merrick , IBM
- T. V. Raman , IBM
- Linda Bucksay Welsh , Intel (*Until April 2001*)
- Gavin McKenzie , JetForm Corporation
- Rob McDougall , JetForm Corporation
- John McCarthy , Lawrence Berkeley National Laboratory (*Until November 2000*)
- Frank Olken , Lawrence Berkeley National Laboratory (*Until November 2000*)
- Ray Waldin , Lexica, LLC
- Tantek Çelik , Microsoft
- Panagiotis Reveliotis , Phillips (*Until December 2000*)
- David Cleary , Progress Software
- Mike Mansell , PureEdge
- Josef Dietl , Mozquito Technologies
- Michael Fergusson , Softquad
- Dave Raggett , W3C/OpenWave (*Until December 2000*)
- Leigh Klotz , Xerox

This document was written with the participation of the XForms Working Group, which currently consists of the following members:

- Steven Pemberton , CWI (*co-chair*)
- Sebastian Schnitzenbaumer , Mozquito Technologies (*co-chair*)
- Micah Dubinko , Cardiff
- Driss Eddaifi , Ecole Mohammadia d'Ingénieurs
- Michalis Petropoulos , Enosys Markets, Inc.
- Peter Stark , Ericsson
- Frank Boumphrey , HTML Writer's Guild
- Roland Merrick , IBM
- T. V. Raman , IBM
- Gavin McKenzie , JetForm Corporation
- Rob McDougall , JetForm Corporation
- Ray Waldin , Lexica, LLC
- Tantek Çelik , Microsoft

- Alex Hopmann , Microsoft
- Dave Hyatt , Netscape/AOL
- Eric Pollmann , Netscape/AOL
- Tom Butcher , OpenDesign
- K. P. Lee , Phillips
- Roli Wendorf , Phillips
- Ted Wugofski , Openwave
- David Cleary , Progress Software
- Mike Mansell , PureEdge
- Dave Manning , PureEdge
- Josef Dietl , Mozquito Technologies
- Michael Fergusson , Softquad
- Zoe Lacroix , SurroMed, Inc.
- Masayasu Ishikawa , W3C
- Leigh Klotz , Xerox

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

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

Note:

Additional Acknowledgments: The editors would like to thank Kai Scheppe, Malte Wedel and Götz Bock for lots of constructive criticism on early versions of the chapter **9 Binding** and their contributions to its present content.

E Production Notes (Non-Normative)

This document was encoded in the XMLspec DTD (which has documentation available). The primary tools used for editing were SoftQuad XMetaL and EMACS with psgml and XAE. The HTML versions were produced with the xmlspec.xsl XSLT stylesheet using the Saxon engine.