



XForms 1.1

W3C Recommendation 20 October 2009

This version:

<http://www.w3.org/TR/2009/REC-xforms-20091020/>

Latest version:

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

Previous version:

<http://www.w3.org/TR/2009/PR-xforms11-20090818/>

Editor:

John M. Boyer, IBM

Please refer to the [errata](#) for this document, which may include normative corrections.

This document is also available in these non-normative formats: [diff-marked HTML](#).

The English version of this specification is the only normative version. Non-normative [translations](#) may also be available.

Copyright © 2009 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use](#) rules apply.

Abstract

XForms is an XML application that represents the next generation of forms for the Web. XForms is not a free-standing document type, but is intended to be integrated into other markup languages, such as XHTML, ODF or SVG. An XForms-based web form gathers and processes XML data using an architecture that separates presentation, purpose and content. The underlying data of a form is organized into **instances** of data schema (though formal schema definitions are not required). An XForm allows processing of data to occur using three mechanisms:

- a declarative **model** composed of formulae for data calculations and constraints, data type and other property declarations, and data submission parameters
- a **view** layer composed of intent-based user interface controls
- an imperative **controller** for orchestrating data manipulations, interactions between the model and view layers, and data submissions.

Thus, XForms accommodates form component reuse, fosters strong data type validation, eliminates unnecessary round-trips to the server, offers device independence and reduces the need for scripting.

XForms 1.1 refines the XML processing platform introduced by [XForms 1.0](#) by adding several new submission capabilities, action handlers, utility functions, user interface improvements, and helpful datatypes as well as a more powerful action processing facility, including conditional, iterated and background execution, the ability to manipulate data arbitrarily and to access event context information.

Status of this Document

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

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

This document was produced by the [W3C Forms Working Group](#) as part of the [Forms Activity](#) within the [W3C Interaction Domain](#). The working group has supplied a [test suite](#) ([zip file](#)) and an [implementation report](#) demonstrating at least one implementation for each test of a feature and at least two interoperable implementations for each test of a required feature.

Please send comments about this document to www-forms-editor@w3.org. (with [public archive](#)). Please send discussion email to www-forms@w3.org (with [public archive](#)).

This document was produced by a group operating under the [5 February 2004 W3C Patent Policy](#). W3C maintains a [public list of any patent disclosures](#) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

Table of Contents

- 1 [About the XForms Specification](#)
 - 1.1 [Background](#)
 - 1.2 [Reading the Specification](#)
 - 1.3 [How the Specification is Organized](#)

- 1.4 [Documentation Conventions](#)
- 1.5 [Differences between XForms 1.1 and XForms 1.0](#)
 - 1.5.1 [Model and Instance](#)
 - 1.5.2 [Enhanced Submissions](#)
 - 1.5.3 [Datatypes and Model Item Properties](#)
 - 1.5.4 [Functions and XPath Expressions](#)
 - 1.5.5 [User Interface](#)
 - 1.5.6 [Actions and Events](#)
- 2 [Introduction to XForms](#)
 - 2.1 [An Example](#)
 - 2.2 [Providing XML Instance Data](#)
 - 2.3 [Constraining Values](#)
 - 2.4 [Multiple Forms per Document](#)
- 3 [Document Structure](#)
 - 3.1 [Namespace for XForms](#)
 - 3.2 [XForms Core Attribute Collections](#)
 - 3.2.1 [Common Attributes](#)
 - 3.2.2 [Linking Attributes](#)
 - 3.2.3 [Single-Node Binding Attributes](#)
 - 3.2.4 [Node-Set Binding Attributes](#)
 - 3.2.5 [Model Item Property Attributes](#)
 - 3.3 [The XForms Core Module](#)
 - 3.3.1 [The model Element](#)
 - 3.3.2 [The instance Element](#)
 - 3.3.3 [The submission Element](#)
 - 3.3.4 [The bind Element](#)
 - 3.4 [The XForms Extension Module](#)
 - 3.4.1 [The extension Element](#)
 - 3.5 [The XForms MustUnderstand Module](#)
- 4 [Processing Model](#)
 - 4.1 [Events Overview](#)
 - 4.2 [Initialization Events](#)
 - 4.2.1 [The xforms-model-construct Event](#)
 - 4.2.2 [The xforms-model-construct-done Event](#)
 - 4.2.3 [The xforms-ready Event](#)
 - 4.2.4 [The xforms-model-destruct Event](#)
 - 4.3 [Interaction Events](#)
 - 4.3.1 [The xforms-rebuild Event](#)
 - 4.3.2 [The xforms-recalculate Event](#)
 - 4.3.3 [The xforms-revalidate Event](#)
 - 4.3.4 [The xforms-refresh Event](#)
 - 4.3.5 [The xforms-reset Event](#)
 - 4.3.6 [The xforms-next and xforms-previous Events](#)
 - 4.3.7 [The xforms-focus Event](#)
 - 4.3.8 [The xforms-help and xforms-hint Events](#)
 - 4.3.9 [The xforms-submit Event](#)
 - 4.3.10 [The xforms-submit-serialize Event](#)
 - 4.4 [Notification Events](#)
 - 4.4.1 [The xforms-insert Event](#)
 - 4.4.2 [The xforms-delete Event](#)
 - 4.4.3 [The xforms-value-changed Event](#)
 - 4.4.4 [The xforms-valid Event](#)
 - 4.4.5 [The xforms-invalid Event](#)
 - 4.4.6 [The xforms-readonly Event](#)
 - 4.4.7 [The xforms-readwrite Event](#)
 - 4.4.8 [The xforms-required Event](#)
 - 4.4.9 [The xforms-optional Event](#)
 - 4.4.10 [The xforms-enabled Event](#)
 - 4.4.11 [The xforms-disabled Event](#)
 - 4.4.12 [The DOMActivate Event](#)
 - 4.4.13 [The DOMFocusIn Event](#)
 - 4.4.14 [The DOMFocusOut Event](#)
 - 4.4.15 [The xforms-select and xforms-deselect Events](#)
 - 4.4.16 [The xforms-in-range Event](#)
 - 4.4.17 [The xforms-out-of-range Event](#)
 - 4.4.18 [The xforms-scroll-first and xforms-scroll-last Events](#)
 - 4.4.19 [The xforms-submit-done Event](#)
 - 4.5 [Error Indications](#)
 - 4.5.1 [The xforms-binding-exception Event](#)
 - 4.5.2 [The xforms-compute-exception Event](#)
 - 4.5.3 [The xforms-version-exception Event](#)
 - 4.5.4 [The xforms-link-exception Event](#)
 - 4.5.5 [The xforms-output-error Event](#)
 - 4.5.6 [The xforms-submit-error Event](#)
 - 4.6 [Event Sequencing](#)
 - 4.6.1 [For input, secret, textarea, range, or upload Controls](#)
 - 4.6.2 [For output Controls](#)
 - 4.6.3 [For select or select1 Controls](#)
 - 4.6.4 [For trigger Controls](#)
 - 4.6.5 [For submit Controls](#)
 - 4.6.6 [Sequence: Selection Without Value Change](#)
 - 4.6.7 [Sequence: Value Change](#)

- 4.6.8 [Sequence: Activating a Trigger](#)
- 4.6.9 [Sequence: Submission](#)
- 4.7 [Resolving ID References in XForms](#)
 - 4.7.1 [References to Elements within a repeat Element](#)
 - 4.7.2 [References to Elements within a bind Element](#)
- 4.8 [DOM Interface for Access to Instance Data](#)
 - 4.8.1 [The getInstanceDocument\(\) Method](#)
 - 4.8.2 [The rebuild\(\) Method](#)
 - 4.8.3 [The recalculate\(\) Method](#)
 - 4.8.4 [The revalidate\(\) Method](#)
 - 4.8.5 [The refresh\(\) Method](#)
- 4.9 [Feature string for the hasFeature method call](#)
- 5 [Datatypes](#)
 - 5.1 [XML Schema Built-in Datatypes](#)
 - 5.2 [XForms Datatypes](#)
 - 5.2.1 [Additional XForms Datatypes to Allow Empty Content](#)
 - 5.2.2 [xforms:listItem](#)
 - 5.2.3 [xforms:listItems](#)
 - 5.2.4 [xforms:dayTimeDuration](#)
 - 5.2.5 [xforms:yearMonthDuration](#)
 - 5.2.6 [xforms:email](#)
 - 5.2.7 [xforms:card-number](#)
- 6 [Model Item Properties](#)
 - 6.1 [Model Item Property Definitions](#)
 - 6.1.1 [The type Property](#)
 - 6.1.2 [The readonly Property](#)
 - 6.1.3 [The required Property](#)
 - 6.1.4 [The relevant Property](#)
 - 6.1.5 [The calculate Property](#)
 - 6.1.6 [The constraint Property](#)
 - 6.1.7 [The p3ptype Property](#)
 - 6.2 [Schema Constraints](#)
 - 6.2.1 [Atomic Datatype](#)
- 7 [XPath Expressions in XForms](#)
 - 7.1 [XPath Datatypes](#)
 - 7.2 [Evaluation Context](#)
 - 7.3 [References, Dependencies, and Dynamic Dependencies](#)
 - 7.4 [Expression Categories](#)
 - 7.4.1 [Model Binding Expressions and Computed Expressions](#)
 - 7.4.2 [Expressions in Actions and Submissions](#)
 - 7.4.3 [UI Expressions](#)
 - 7.4.4 [UI Binding in other XML vocabularies](#)
 - 7.4.5 [Binding Examples](#)
 - 7.5 [The XForms Function Library](#)
 - 7.6 [Boolean Functions](#)
 - 7.6.1 [The boolean-from-string\(\) Function](#)
 - 7.6.2 [The is-card-number\(\) Function](#)
 - 7.7 [Number Functions](#)
 - 7.7.1 [The avg\(\) Function](#)
 - 7.7.2 [The min\(\) Function](#)
 - 7.7.3 [The max\(\) Function](#)
 - 7.7.4 [The count-non-empty\(\) Function](#)
 - 7.7.5 [The index\(\) Function](#)
 - 7.7.6 [The power\(\) Function](#)
 - 7.7.7 [The random\(\) Function](#)
 - 7.7.8 [The compare\(\) Function](#)
 - 7.8 [String Functions](#)
 - 7.8.1 [The if\(\) Function](#)
 - 7.8.2 [The property\(\) Function](#)
 - 7.8.3 [The digest\(\) Function](#)
 - 7.8.4 [The hmac\(\) Function](#)
 - 7.9 [Date and Time Functions](#)
 - 7.9.1 [The local-date\(\) Function](#)
 - 7.9.2 [The local-dateTime\(\) Function](#)
 - 7.9.3 [The now\(\) Function](#)
 - 7.9.4 [The days-from-date\(\) Function](#)
 - 7.9.5 [The days-to-date\(\) Function](#)
 - 7.9.6 [The seconds-from-dateTime\(\) Function](#)
 - 7.9.7 [The seconds-to-dateTime\(\) Function](#)
 - 7.9.8 [The adjust-dateTime-to-timezone\(\) Function](#)
 - 7.9.9 [The seconds\(\) Function](#)
 - 7.9.10 [The months\(\) Function](#)
 - 7.10 [Node-set Functions](#)
 - 7.10.1 [The instance\(\) Function](#)
 - 7.10.2 [The current\(\) Function](#)
 - 7.10.3 [The id\(\) Function](#)
 - 7.10.4 [The context\(\) Function](#)
 - 7.11 [Object Functions](#)
 - 7.11.1 [The choose\(\) Function](#)
 - 7.11.2 [The event\(\) Function](#)
 - 7.12 [Extension Functions](#)
- 8 [Core Form Controls](#)

- 8.1 [The XForms Core Form Controls Module](#)
 - 8.1.1 [Implementation Requirements Common to All Form Controls](#)
 - 8.1.2 [The input Element](#)
 - 8.1.3 [The secret Element](#)
 - 8.1.4 [The textarea Element](#)
 - 8.1.5 [The output Element](#)
 - 8.1.5.1 [The mediatype Element \(for output\)](#)
 - 8.1.6 [The upload Element](#)
 - 8.1.6.1 [The filename Element](#)
 - 8.1.6.2 [The mediatype Element \(for upload\)](#)
 - 8.1.7 [The range Element](#)
 - 8.1.8 [The trigger Element](#)
 - 8.1.9 [The submit Element](#)
 - 8.1.10 [The select Element](#)
 - 8.1.11 [The select1 Element](#)
- 8.2 [Common Support Elements](#)
 - 8.2.1 [The label Element](#)
 - 8.2.2 [The help Element](#)
 - 8.2.3 [The hint Element](#)
 - 8.2.4 [The alert Element](#)
- 8.3 [Common Markup for Selection Controls](#)
 - 8.3.1 [The choices Element](#)
 - 8.3.2 [The item Element](#)
 - 8.3.3 [The value Element](#)
- 9 [Container Form Controls](#)
 - 9.1 [The XForms Group Module](#)
 - 9.1.1 [The group Element](#)
 - 9.2 [The XForms Switch Module](#)
 - 9.2.1 [The switch Element](#)
 - 9.2.2 [The case Element](#)
 - 9.3 [The XForms Repeat Module](#)
 - 9.3.1 [The repeat Element](#)
 - 9.3.2 [Nested Repeats](#)
 - 9.3.3 [Repeat Processing](#)
 - 9.3.4 [User Interface Interaction](#)
 - 9.3.5 [Creating Repeating Structures Via Attributes](#)
 - 9.3.6 [The itemset Element](#)
 - 9.3.7 [The copy Element](#)
- 10 [XForms Actions](#)
 - 10.1 [The action Element](#)
 - 10.2 [The setvalue Element](#)
 - 10.3 [The insert Element](#)
 - 10.4 [The delete Element](#)
 - 10.5 [The setindex Element](#)
 - 10.6 [The toggle Element](#)
 - 10.6.1 [The case Element Child of the toggle Element](#)
 - 10.7 [The setfocus Element](#)
 - 10.7.1 [The control Element Child of the setfocus Element](#)
 - 10.8 [The dispatch Element](#)
 - 10.8.1 [The name Child Element](#)
 - 10.8.2 [The targetid Child Element](#)
 - 10.8.3 [The delay Child Element](#)
 - 10.9 [The rebuild Element](#)
 - 10.10 [The recalculate Element](#)
 - 10.11 [The revalidate Element](#)
 - 10.12 [The refresh Element](#)
 - 10.13 [The reset Element](#)
 - 10.14 [The load Element](#)
 - 10.14.1 [The resource Element child of load](#)
 - 10.15 [The send Element](#)
 - 10.16 [The message Element](#)
 - 10.17 [Conditional Execution of XForms Actions](#)
 - 10.18 [Iteration of XForms Actions](#)
 - 10.19 [Actions from Other Modules](#)
- 11 [The XForms Submission Module](#)
 - 11.1 [The submission Element](#)
 - 11.2 [The xforms-submit Event](#)
 - 11.3 [The xforms-submit-serialize Event](#)
 - 11.4 [The xforms-submit-done Event](#)
 - 11.5 [The xforms-submit-error Event](#)
 - 11.6 [The Submission Resource](#)
 - 11.6.1 [The resource Element](#)
 - 11.7 [The Submission Method](#)
 - 11.7.1 [The method Element](#)
 - 11.8 [The header Element](#)
 - 11.8.1 [The name Element](#)
 - 11.8.2 [The value Element](#)
 - 11.9 [Submission Options](#)
 - 11.9.1 [The get Submission Method](#)
 - 11.9.2 [The post, multipart-post, form-data-post, and urlencoded-post Submission Methods](#)
 - 11.9.3 [The put Submission Method](#)
 - 11.9.4 [The delete Submission Method](#)

- 11.9.5 [Serialization as application/xml](#)
- 11.9.6 [Serialization as multipart/related](#)
- 11.9.7 [Serialization as multipart/form-data](#)
- 11.9.8 [Serialization as application/x-www-form-urlencoded](#)
- 11.10 [Replacing Data with the Submission Response](#)
- 11.11 [Integration with SOAP](#)
 - 11.11.1 [Representation of SOAP Envelope](#)
 - 11.11.2 [Indicating a SOAP submission](#)
 - 11.11.3 [SOAP HTTP Binding](#)
 - 11.11.4 [Handling the SOAP Response](#)
- 12 [Conformance](#)
 - 12.1 [Conforming XForms Documents](#)
 - 12.2 [Conforming XForms Generators](#)
 - 12.3 [Base Technologies for XForms Processors](#)
 - 12.4 [Conformance Levels](#)
 - 12.4.1 [XForms Model](#)
 - 12.4.2 [XForms Full](#)
- 13 [Glossary Of Terms](#)

Appendices

- A [References](#)
 - A.1 [Normative References](#)
 - A.2 [Informative References](#)
- B [Patterns for Data Mutations](#)
 - B.1 [Prepend Element Copy](#)
 - B.2 [Append Element Copy](#)
 - B.3 [Duplicate Element](#)
 - B.4 [Set Attribute](#)
 - B.5 [Remove Element](#)
 - B.6 [Remove Attribute](#)
 - B.7 [Remove Nodeset](#)
 - B.8 [Copy Nodeset](#)
 - B.9 [Copy Attribute List](#)
 - B.10 [Replace Element](#)
 - B.11 [Replace Attribute](#)
 - B.12 [Replace Instance with Insert](#)
 - B.13 [Move Element](#)
 - B.14 [Move Attribute](#)
 - B.15 [Insert Element into Non-Contiguous, Heterogeneous Nodeset](#)
- C [Recalculation Sequence Algorithm](#)
 - C.1 [Details on Creating the Master Dependency Directed Graph](#)
 - C.2 [Details on Creating the Pertinent Dependency Subgraph](#)
 - C.3 [Details on Computing Individual Vertices](#)
 - C.4 [Example of Calculation Processing](#)
- D [Privacy Considerations](#)
 - D.1 [Using P3P with XForms](#)
- E [Input Modes](#) (Non-Normative)
 - E.1 [inputmode Attribute Value Syntax](#)
 - E.2 [User Agent Behavior](#)
 - E.3 [List of Tokens](#)
 - E.3.1 [Script Tokens](#)
 - E.3.2 [Modifier Tokens](#)
 - E.4 [Relationship to XML Schema pattern facets](#)
 - E.5 [Examples](#)
- F [Schema for XForms](#) (Non-Normative)
 - F.1 [Schema for XML Events](#)
- G [XForms and Styling](#) (Non-Normative)
 - G.1 [Pseudo-classes](#)
 - G.2 [Pseudo-elements](#)
 - G.3 [Examples](#)
- H [Complete XForms Examples](#) (Non-Normative)
 - H.1 [XForms in XHTML](#)
 - H.2 [Editing Hierarchical Bookmarks Using XForms](#)
 - H.3 [Survey Using XForms and SVG](#)
- I [Acknowledgements](#) (Non-Normative)
- J [Production Notes](#) (Non-Normative)

1 About the XForms Specification

1.1 Background

Forms are an important part of the Web, and they continue to be the primary means for enabling interactive Web applications. Web applications and electronic commerce solutions have sparked the demand for better Web forms with richer interactions. XForms is the response to this demand, and provides a new platform-independent markup language for online interaction between a person (through an [XForms Processor](#)) and another, usually remote, agent. XForms are the successor to HTML forms, and benefit from the lessons learned from HTML forms.

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

1.2 Reading the Specification

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

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

With regard to implementing behaviors defined for XForms content herein, this document uses the terms **must**, **must not**, **required**, **shall**, **shall not**, **recommended**, **should**, **should not**, **may**, and **optional** in accord with [\[RFC 2119\]](#). Generally, the elements, attributes, functions and behaviors of XForms content defined in this specification are required to implement unless explicitly specified otherwise. The term **author-optional**, when applied to a content item such as an element, attribute, or function parameter, indicates to form authors that they may omit the content item and obtain the default behavior. The term author-optional is orthogonal to the conformance status (required, recommended, or optional) of the content item.

1.3 How the Specification is Organized

The specification is organized into the following chapters:

Chapters 1 and 2

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

Chapters 3 and up

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

Appendixes

Appendixes contain an XML Schema description of XForms, references, examples, and other useful information.

1.4 Documentation Conventions

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

- xforms**: The XForms namespace, e.g. <http://www.w3.org/2002/xforms> (see [3.1 Namespace for XForms](#))
- html**: An XHTML namespace, e.g. <http://www.w3.org/1999/xhtml> (see [\[XHTML 1.0\]](#))
- xs**: The XML Schema namespace <http://www.w3.org/2001/XMLSchema> (see [\[XML Schema part 1\]](#))
- xsd**: The XML Schema namespace <http://www.w3.org/2001/XMLSchema> (see [\[XML Schema part 2\]](#))
- xsi**: The XML Schema for instances namespace <http://www.w3.org/2001/XMLSchema-instance> (see [\[XML Schema part 1\]](#))
- ev**: The XML Events namespace <http://www.w3.org/2001/xml-events> (see [\[XML Events\]](#))
- my**: Any user defined namespace

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

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

Official terms are defined in the following manner: [Definition: You can find most **terms** in chapter [13 Glossary Of Terms](#)]. Links to **terms** may be specially highlighted where necessary.

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

Examples are set off typographically:

Example item
Example Item

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

Sample Reference

Reference - linked to from above.

The following typesetting convention is used for additional commentary:

Note:

A gentle explanation to readers.

Editorial note: Editorial Note Name	
Editorial commentary, not intended for final publication.	

Issue (sample-implementation-issue):

Issue-Name

A specific issue for which input from implementors is requested, for example as part of the Candidate Recommendation phase.

Resolution:

None recorded.

1.5 Differences between XForms 1.1 and XForms 1.0

This informative section provides an overview of the new features and changed behaviors available in XForms 1.1.

1.5.1 Model and Instance

The `model` element now must support a `version` attribute to help authors bridge the transition between XForms 1.0 to XForms 1.1.

The `instance` element now has a `resource` attribute that allows instance data to be obtained from a URI only if the instance does not already contain data. By contrast, the `src` attribute overrides the inline content in an `instance`. The `resource` attribute is more useful in systems that must support save and reload of XForms-based documents.

1.5.2 Enhanced Submissions

The `submission` element offers many new features that allow significantly improved data communications capabilities for XForms, including:

- Access to SOAP-based web services, RESTful services, ATOM-based services, and non-XML services
- Improved control over submission processing and serialization
- Ability to control the submission URI and headers with instance data
- Targetted instance data replacement capabilities

The `submission` element now has a `resource` attribute and `resource` child element that allow the instance data to dynamically control the submission URI. As a result, the `action` attribute is deprecated, though still supported in XForms 1.1.

In XForms 1.0, submissions were already more capable than AJAX, based on the ability to automatically update a form with results from HTTP and HTTPS services, including RSS feeds. In XForms 1.1, the `method` attribute now supports `delete` as well as any other QName. The `method` child element also allows the method to be dynamically controlled by instance data. Submission headers can now be added, and even dynamically controlled by instance data, using the `header` child element. These features complete the capabilities needed for ATOM and RESTful services. XForms 1.1 also offers special submission header behavior through the `mediatype` attribute to allow communications with SOAP 1.1 and 1.2 web services.

The `submission` element now supports attributes `relevant` and `validate`, which allow form authors to turn off instance data relevance pruning and validity checking. This allows `submission` to be used to save and reload unfinished data on a server or the local file system.

The `submission` element now supports the `targetref` attribute, which allows partial instance replacement by identifying a node to be replaced with the submission result. The `replace` attribute also now supports a `text` setting, which allows the content of the target node, rather than the target node itself, to be replaced with a non-XML (text) submission result.

The `submission` element now also supports the `xforms-submit-serialize` event, which allows the form author to provide a custom serialization, such as plain text or the full XForms document, as the submission data. The `serialization` attribute also provides increased control over the submission data serialization, including the setting `none`, which allows `submission` to be used for simple URI activation.

The `xforms-submit-done` and `xforms-submit-error` events now have event context information available that provide more information about both successful and failed submissions, such as the response headers of successful submissions and the reason code for failed submissions.

Finally, over a dozen new examples have been added to illustrate `submission` usage.

1.5.3 Datatypes and Model Item Properties

XForms 1.1 now offers `email` and `card-number` datatypes so form authors can easily validate email address and credit card number input values.

To further simplify authoring, XForms 1.1 now also provides its own definitions of the XML Schema datatypes, except the XForms versions permit the empty string. Allowing empty string means that input like an age or a birthdate can be collected without being required input for validity (an empty string is not in the lexical space of XML schema datatypes like `xsd:positiveInteger` and `xsd:date`). If an input is required, the form author can still use the XForms versions of the datatypes in combination with the `required` model item property. The XForms datatypes also aid authoring by allowing type definitions to omit namespace qualification, e.g. `type="date"` rather than `type="xsd:date"`, if the default namespace of the model is set to XForms.

The `readonly` model item property was defined to be an inviolate property of the data model. This means it cannot be violated by anything outside of the model item property system, including not just form controls but also XForms actions and instance data access from the DOM interface.

1.5.4 Functions and XPath Expressions

XForms 1.1 now contains many new functions that can be used in `calculate` and other XPath expressions to enable numerous features, including:

- basic date math and working with local dates and times: `local-date()`, `local-dateTime()`, `days-to-date()`, `seconds-to-dateTime()`, and `adjust-dateTime-to-timezone()`
- working with tabular data and parallel lists: `current()`, `choose()` and `context()`
- basic security capabilities: `digest()`, `hmac()`, and `random()`
- improved numeric and string processing: `power()`, `is-card-number()`, and `compare()`
- search across instances of a model: two parameter `id()` function

- access to context information added to many XForms events: `event()`

The specification now provides a better classification of binding expression types as well as a more rigorous definition for dynamic dependencies. These definitions ensure that XPath expressions in form controls and actions which use the `index()` are automatically re-evaluated when appropriate.

Due to the addition of the `choose()` function, the `if()` function is still supported but deprecated as futureproofing against the conflict with the `if` keyword in XPath 2.0.

1.5.5 User Interface

The behavioral description common to all form controls has been improved to indicate default layout styling and rendering requirements for required data.

The `output` form control has been improved to render non-text mediatypes, particularly images, obtained from instance data.

An example was added to show the use of a `DOMActivate` handler on an `input` to automatically initiate a submission once a user enters and commits input, such as a search query.

The processing model and implementation requirements on selection controls were elaborated upon to ensure consistency of behavior between selection data expressed as textual lists versus element lists.

The ability to create wizard-like interfaces with dynamically available form controls has been improved. Details are in the description of improvements to actions.

The specification provides more rigorous definitions and classifications of form controls, which have been applied throughout the specification to ensure proper support of varied features related to form controls, such as events, applicability of model item properties, and focusability.

The XForms repeat has been made more powerful and flexible. The specification now provides rigorous definitions and processing model descriptions for repeated content, including creation, destruction, IDREF resolution and event flow between repeated content and the containing content (which may itself be repeated). The `repeat` is now capable of operating over any nodeset, not just an homogeneous collection. A formal processing model for repeat index handling has been defined.

1.5.6 Actions and Events

The `insert` and `delete` actions have been converted from specialized actions associated with `repeat` to generalized data insertion and deletion operations. An entire appendix of 15 examples was added to illustrate this additional capability in detail.

All XForms actions, as well as sets of actions, can be executed conditionally or iteratively. Combined with the generalized `insert` and `delete`, this means that the information processing power of XForms 1.1 is Turing-complete.

The `dispatch` action now allows the event name and target to be specified by instance data. A new attribute, `delay`, has also been added to allow an event to be scheduled for dispatch at a later time. Since the event handler for the event can schedule same event for later dispatch, it is possible in XForms 1.1 to create background daemon tasks.

The `setfocus` and `toggle` have been improved to help with creating wizard interfaces and handling dynamically available content. The control to focus and the case to select can now be specified by instance data. These actions have also been improved relative to the recalculation processing model. They now perform deferred updates before their regular processing to ensure the user interface is automatically refreshed.

As part of the improvement to repeat index management, the `setindex` action now behaves more like `setvalue`, which means it now sets the flags for automatic recalculation, revalidation and user interface refresh. As well, this action now also performs deferred updates before its regular processing to ensure the user interface is up to date.

Finally, the `setvalue` action has been improved due to the addition of the `context()` function. Now it is possible to express the `value` attribute in terms of the same context node used to evaluate the single node binding. This improves the ability to use `setvalue` inside of a `repeat` to set values of instance nodes that are outside of the repeat nodeset based on values that are within the repeat nodeset.

2 Introduction to XForms

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

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

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

XForms has striven to improve authoring, reuse, internationalization, accessibility, usability, and device independence. Here is a summary of the primary benefits of using XForms:

Strong typing

Submitted data is strongly typed and can be checked using off-the-shelf tools. This speeds up form filling since it reduces the need for round trips to the server for validation.

XML submission

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

Existing schema re-use

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

External schema augmentation

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

Internationalization

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

Enhanced accessibility

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

Multiple device support

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

Less use of scripting

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

2.1 An Example

In the XForms approach, forms are comprised of a section that describes what the form does, called the [XForms Model](#), and another section that describes how the form is to be presented.

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

Select Payment Method: ☒ Cash ☐ Credit

Credit Card Number:

Expiration Date:

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

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

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

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

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

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

Notice the following features of this design:

- The user interface is not hard-coded to use radio buttons. Different devices (such as voice browsers) can render the concept of "select one" as appropriate.
- [Core form controls](#) always have labels directly associated with them as child elements— this is a key feature designed to enhance accessibility.
- There is no need for an enclosing `form` element, as in HTML. (See [2.4 Multiple Forms per Document](#) for details on how to author multiple forms per document)
- Markup for specifying form controls has been simplified in comparison with HTML forms.

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

2.2 Providing XML Instance Data

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

Submitted Data

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

XForms processing keeps track of the state of the partially filled form through this [instance data](#). Initial values for the instance data may be provided or left empty as in the example. Element `instance` essentially holds a skeleton XML document that gets updated as the user fills out the form. It gives the author full control on the structure of the submitted XML data, including namespace information. When the form is submitted, the instance data is serialized as an XML document. Here is an alternative version of the earlier example:

Model

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

In this case the submitted data would look like this:

Submitted Data

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

This design has features worth calling out:

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

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

Binding Form Controls to Instance Nodes with `ref`

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

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

2.3 Constraining Values

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

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

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

Declarative Validation with Model Item Properties

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

<xforms:model>
  ...
  <xforms:bind nodeset="/my:payment/my:number"
    relevant="/my:payment/@method = 'cc'"
    required="true()"
    type="my:ccnumber"/>

  <xforms:bind nodeset="/my:payment/my:expiry"
    relevant="/my:payment/@method = 'cc'"
    required="true()"
    type="xsd:gYearMonth"/>

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

Note:

In the above example, the `relevant` expression uses absolute XPath notation (beginning with `/`) because the evaluation context nodes for [computed expressions](#) are determined by the [binding expression](#) (see [7.2 Evaluation Context](#)), and so any relative node path in the first `bind relevant` above would be relative to `/my:payment/my:number`

2.4 Multiple Forms per Document

XForms processing places no limits on the number of individual forms that can be placed in a single [containing document](#). When a single document contains multiple forms, each form needs a separate `model` element, each with an `id` attribute so that they can be referenced from elsewhere in the containing document.

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

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

Adding a `poll` model

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

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

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

Form Controls for `poll` model

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

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

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

More XForms examples can be found in [H Complete XForms Examples](#).

3 Document Structure

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

3.1 Namespace for XForms

The namespace URI for XForms is `http://www.w3.org/2002/xforms`. The XForms schema has the target namespace specified and as such is compatible with the XForms 1.0 definition.

XForms used in combination with XHTML 1.0

```
<switch xmlns="http://www.w3.org/2002/xforms">
  <case id="in" selected="true">
    <input ref="yourname">
      <label>Please tell me your name</label>
      <toggle ev:event="DOMActivate" case="out"/>
    </input>
  </case>
  <case id="out" selected="false">
    <html:p>Hello <output ref="yourname" />
    <trigger id="editButton">
      <label>Edit</label>
      <toggle ev:event="DOMActivate" case="in"/>
    </trigger>
  </html:p>
</case>
</switch>
```

The above example is unchanged from the specification in XForms 1.0 (in the example, the prefixes `html` and `ev` are defined by an ancestor of the `switch` element).

3.2 XForms Core Attribute Collections

3.2.1 Common Attributes

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

anyAttribute

Foreign attributes are allowed on all XForms elements.

id

The author-optional `id` attribute of type `xsd:ID` assigns an identity to the containing element.

Note:

Elements can be identified using any attribute of type ID (such as `xml:id`), not just the `id` attribute defined above.

Note:

The [XML Events attributes](#) are foreign attributes and therefore are allowed on any XForms element that includes the [Common](#) attributes. This specification lists both [Common](#) and [Events](#) attributes on XForms actions for reading convenience, i.e. since authors are most likely to place [Events](#) attributes on the actual event handler elements.

3.2.2 Linking Attributes

The [host language](#) may permit a Linking Attributes Collection to be applied to XForms elements as an alternate means of obtaining content related to the element. An example is the `src` attribute from [\[XHTML 1.0\]](#). The schedule by which link traversal occurs is defined by the host language. If the link traversal fails, the host language may dispatch [xforms-link-exception](#) to the `model` associated with the in-scope evaluation context node of the element that bears the Linking Attributes Collection for the failed link.

Note:

Section [3.3.2 The instance Element](#) defines attribute `src` for the `instance` element.

3.2.3 Single-Node Binding Attributes

The following attributes can be used to define a binding between an XForms element such as a form control or an action and an instance data node defined by an XPath expression.

ref

[Binding expression](#) interpreted as XPath. This attribute has no meaning when a `bind` attribute is present.

model

Author-optional XForms Model selector. Specifies the `ID` of an XForms Model to be associated with this binding element. This attribute has no meaning for the current binding element when a `bind` attribute is present. Rules for determining the context XForms Model are located at [7.2 Evaluation Context](#).

bind

Author-optional reference to a `bind` element.

In this specification, when an XForms element is declared to have a Single-Node Binding, then the author must specify the Single-Node Binding unless the element explicitly states that it is author-optional.

In some cases, an XForms element may allow a Single-Node Binding, but one or more attributes in the Single-Node Binding attribute group are inappropriate for that XForms element. In such cases, the exact attributes are listed for the XForms element, but those attributes still express a Single-Node Binding if they appear in the element. For example, the `submission` element forbids the `model` attribute because the model is defined to be the one containing the `submission`, so the attributes `ref` and `bind` are listed for `submission` rather than referring to the Single-Node Binding attribute group, but if a `ref` or `bind` attribute is used on a `submission`, it does express a Single-Node Binding.

When the Single-Node Binding is required, one of `ref` or `bind` is required. When `bind` is used, the node is determined by the referenced `bind`. See [4.7.2 References to Elements within a bind Element](#) for details on selecting an identified `bind` that is iterated by one or more containing `bind` elements. When `ref` is used, the node is determined by evaluating the XPath expression with the evaluation context described in Section [7.2 Evaluation Context](#).

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

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

3.2.4 Node-Set Binding Attributes

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

nodeset

[Binding expression](#) interpreted as XPath. This attribute has no meaning when a `bind` attribute is present.

model

Author-optional XForms Model selector. Specifies the `ID` of an XForms Model to be associated with this binding element. This attribute has no meaning for the current binding element when a `bind` attribute is present. Rules for determining the context XForms Model are located at [7.2 Evaluation Context](#).

bind

Author-optional reference to a `bind` element.

In this specification, when an XForms element is declared to have a Node-Set Binding, then the author must specify the Node-Set Binding unless the element explicitly states that it is author-optional.

In some cases, an XForms element may allow a Node-Set Binding, but one or more attributes in the Node-Set Binding attribute group are inappropriate for that XForms element. In such cases, the exact attributes are listed for the XForms element, but those attributes still express a Node-Set Binding if they appear in the element. For example, the `bind` element only allows the `nodeset` attribute. The `model` and `bind` attributes are not allowed on a `bind` element, but if the `nodeset` attribute appears on a `bind` element, it does express a Node-Set Binding.

When the Node-Set Binding is required, one of `nodeset` or `bind` is required. When `bind` is used, the node-set is determined by the referenced `bind`. See [4.7.2 References to Elements within a bind Element](#) for details on selecting an identified `bind` that is iterated by one or more containing `bind` elements. When `nodeset` is used, the node-set is determined by evaluating the XPath expression with the evaluation context described in Section [7.2 Evaluation Context](#).

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

3.2.5 Model Item Property Attributes

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

3.3 The XForms Core Module

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

Element	Attributes	Minimal Content Model
model	Common , functions (QNameList), schema (list of <code>xsd:anyURI</code>), version (xforms:versionList)	(instance xs:schema submission bind Action)*
instance	Common , src (<code>xsd:anyURI</code>), resource (<code>xsd:anyURI</code>)	(ANY)
	Common	

submission	ref (binding-expression) bind (xsd:IDREF) resource (xsd:anyURI) action (xsd:anyURI) [deprecated] mode ("asynchronous" "synchronous") method ("post" "get" "put" "delete" "multipart-post" "form-data-post" "urlencoded-post" "Any other NCName QNameButNotNCName) validate (xsd:boolean) relevant (xsd:boolean) serialization ("application/xml" "application/x-www-form-urlencoded" "multipart/related" "multipart/form-data" "none") version (xsd:NMTOKEN) indent (xsd:boolean) mediatype (xsd:string) encoding (xsd:string) omit-xml-declaration (xsd:boolean) standalone (xsd:boolean) cdata-section-elements (QNameList) replace ("all" "instance" "text" "none" QNameButNotNCName) instance (xsd:IDREF) targetref (nodeset XPath Expression) separator (';' '&') includenamespaceprefixes (xsd:NMTOKENS)	(resource method header)*, Action*
bind	Common , Model Item Properties , nodeset (model-binding-expression)	(bind)*

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

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

The [XForms Processor](#) must ignore any foreign-namespaced attributes that are unrecognized.

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

3.3.1 The model Element

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

Common Attributes: [Common](#)

Special Attributes:

functions

Author-optional space-separated list of XPath extension functions (represented by QNames) required by this XForms Model. Guidance on the use of this attribute is at [7.12 Extension Functions](#).

schema

Author-optional list of `xsd:anyURI` links to XML Schema documents outside this `model` element. The [XForms Processor](#) must process all Schemas listed in this attribute. Within each XForms Model, there is a limit of one Schema per namespace declaration, including inline and linked Schemas.

The schema definitions for a namespace are determined to be **applicable** to instance nodes based on an instance schema validation episode initialized to [lax processing](#). When an element lacks a schema declaration, the XML Schema specification defines the recursive checking of children and attributes as optional. For this specification, this recursive checking is required. Schema processing for a node with matching schema declarations is governed by its content processing definition, which is [strict](#) by default.

Note:

The `schema` list may include URI fragments referring to elements located outside the current model elsewhere in the containing document; e.g. `#myschema`. `xs:schema` elements located inside the current model need not be listed.

version

Author-optional attribute with a default value of empty string and legal values defined by the datatype [xforms:versionList](#). Examples are "1.0" and "1.0 1.1". If one or more versions are indicated by this attribute on the default `model`, then an [XForms Processor](#) must support at least one of the listed language versions of XForms. Otherwise, the [XForms Processor](#) must terminate processing after dispatching the event [xforms-version-exception](#) to the default `model`. If the [XForms Processor](#) supports more than one language version indicated by the version setting on the default `model` or if the version setting on the default `model` is empty string (whether specified or by default), then the [XForms Processor](#) may execute the XForms content using any language conformance level available to it. If any non-default `model` has a version setting that is incompatible with the language version selected by the [XForms Processor](#), then the [XForms Processor](#) must terminate processing after dispatching the event [xforms-version-exception](#) to the default `model`.

Examples:

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

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

Handler for xforms-version-exception

```

<model>
  <message level="modal" ev:event="xforms-version-exception">
    <output value="event('errorinformation')"/>
  </message>
  ...
</model>
...
<model id="m2" version="1.1">
  ...
</model>

```

Since the `version` attribute is not specified on the `model`, the [XForms Processor](#) may choose any language conformance level, which may be incompatible with the version setting of the second `model`. Therefore, the message action occurs during initialization of the second `model` due to its version incompatibility with the default `model`.

An Example of Differing but Compatible Version Settings

```

<model version="1.0 1.1">
  ...
</model>
...
<model id="m2">
  ...
</model>

```

Since the `version` attribute is not specified on the second `model`, it is compatible with any choice made based on the version setting on the default `model`.

3.3.2 The instance Element

This author-optional element contains or references initial instance data.

Common Attributes: [Common](#)

Special Attributes:

src

Author-optional link to externally defined initial instance data. If the link traversal fails, it is treated as an exception ([4.5.4 The xforms-link-exception Event](#)).

resource

Author-optional link to externally defined initial instance data. If the link is traversed and the traversal fails, it is treated as an exception ([4.5.4 The xforms-link-exception Event](#)).

If the `src` attribute is given, then it takes precedence over inline content and the `resource` attribute, and the XML data for the instance is obtained from the link. If the `src` attribute is omitted, then the data for the instance is obtained from inline content if it is given or the `resource` attribute otherwise. If both the `resource` attribute and inline content are provided, the inline content takes precedence.

If the initial instance data is given by a link (from `src` or `resource`), then the instance data is formed by creating an XPath data model of the linked resource. If the link cannot be traversed, then processing halts after dispatching an [xforms-link-exception](#) with a `resource-uri` of the link that failed.

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

If creation of the XPath data model for the instance data fails due to an XML error, then processing halts after dispatching an [xforms-link-exception](#) with a `resource-uri` indicating either the URI for an external instance, a fragment identifier URI reference (including the leading # mark) for an identified internal instance, or empty string for an unidentified internal instance. This exception could happen, for example, if the content had no top-level element or more than one top-level element, neither of which is permitted by the grammar of XML.

Note:

All data relevant to the XPath data model must be preserved during processing and as input to submission serialization, including processing instructions, comment nodes and all whitespace.

Note:

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

3.3.3 The submission Element

Details about the `submission` element and its processing are described in [11 The XForms Submission Module](#).

3.3.4 The bind Element

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

Common Attributes: [Common](#), [Model Item Properties](#) (author-optional)

Special Attributes:

nodeset

An author-optional attribute containing a [model binding expression](#) that selects the set of nodes on which this `bind` operates. If the attribute is omitted, the default is the in-scope evaluation context node.

See [6 Model Item Properties](#) for details on model item properties.

See [7.2 Evaluation Context](#) for details on how the evaluation context is determined for each attribute of the `bind` element.

3.4 The XForms Extension Module

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

Element	Attributes	Minimal Content Model
extension	Common	ANY

3.4.1 The extension Element

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

Common Attributes: [Common](#)

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

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

3.5 The XForms MustUnderstand Module

This section is deleted.

4 Processing Model

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

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

- Initialization
- Interaction
- Notification
- Error Conditions

4.1 Events Overview

XForms processing is defined in terms of events, event handlers, and event responses. XForms uses the events system defined in [\[DOM2 Events\]\[XML_Events\]](#), with an event capture phase, arrival of the event at its Target, and finally the event bubbling phase.

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

xforms-revalidate	Yes	Yes	model
xforms-refresh	Yes	Yes	model
xforms-reset	Yes	Yes	model
xforms-previous	Yes	No	Core Form Controls
xforms-next	Yes	No	Core Form Controls
xforms-focus	Yes	No	Core Form Controls groupswitchrepeat
xforms-help	Yes	Yes	Core Form Controls
xforms-hint	Yes	Yes	Core Form Controls
xforms-submit	Yes	Yes	submission
xforms-submit-serialize	No	Yes	submission
4.4 Notification Events			
xforms-insert	No	Yes	instance
xforms-delete	No	Yes	instance
xforms-value-changed	No	Yes	Core Form Controls
xforms-valid	No	Yes	Core Form Controls groupswitch
xforms-invalid	No	Yes	Core Form Controls groupswitch
xforms-readonly	No	Yes	Core Form Controls groupswitch
xforms-readwrite	No	Yes	Core Form Controls groupswitch
xforms-required	No	Yes	Core Form Controls groupswitch
xforms-optional	No	Yes	Core Form Controls groupswitch
xforms-enabled	No	Yes	Core Form Controls groupswitch
xforms-disabled	No	Yes	Core Form Controls groupswitch
DOMActivate	Yes	Yes	Core Form Controls
DOMFocusIn	No	Yes	Core Form Controls groupswitchrepeat
DOMFocusOut	No	Yes	Core Form Controls groupswitchrepeat
xforms-select	No	Yes	item or case
xforms-deselect	No	Yes	item or case
xforms-in-range	No	Yes	Core Form Controls
xforms-out-of-range	No	Yes	Core Form Controls
xforms-scroll-first	No	Yes	repeat
xforms-scroll-last	No	Yes	repeat
xforms-submit-done	No	Yes	submission
4.5 Error Indications			
xforms-binding-exception	No	Yes	any element that can contain a binding expression
xforms-compute-exception	No	Yes	model
xforms-version-exception	No	Yes	The default model
xforms-link-exception	No	Yes	model
xforms-output-error	No	Yes	output
xforms-submit-error	No	Yes	submission

4.2 Initialization Events

This section defines the various stages of the *initialization* phase. The processor begins initialization by dispatching an event `xforms-model-construct` to each XForms Model in the containing document. How the XForms Processor itself is requested to initialize is implementation dependent.

4.2.1 The xforms-model-construct Event

Dispatched to each XForms model by the XForms processor.

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

The default action for this event results in the following:

1. All XML Schemas are loaded. If an error occurs while attempting to access or process a remote document, processing halts with an exception ([4.5.4 The xforms-link-exception Event](#)).
2. For each `instance` element, an XPath data model [[7 XPath Expressions in XForms](#)] is constructed from it as described in Section [3.3.2 The instance Element](#). If there are no `instance` elements, the data model is not constructed in this phase, but during user interface construction ([4.2.2 The xforms-model-construct-done Event](#)).
3. If applicable, P3P initialization occurs. [[P3P 1.0](#)]
4. Perform the behaviors of `xforms-rebuild`, `xforms-recalculate`, and `xforms-revalidate` in sequence on this `model` element without

dispatching events to invoke the behaviors. The notification event markings for these operations are discarded, and the `xforms-refresh` behavior is not performed since the user interface has not yet been initialized.

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

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

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

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

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

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

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

1. The single node binding expression is evaluated, if it exists on the form control, to ensure that it points to a node that exists. If this is not the case then the form control should behave in the same manner as if it had bound to a model item with the `relevant` model item property resolved to `false`.
2. Otherwise, the user interface for the form control is created and initialized.

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

1. For the first reference to an `instance` a default `instance` is created by following the rules described below.
 - a. A root `instanceData` element is created.
 - b. An instance data element node will be created using the binding expression from the user interface control as the `name`. If the `name` is not a valid QName, processing halts with an exception ([4.5.1 The `xforms-binding-exception` Event](#)).
2. For the second and subsequent references to an `instance` which was automatically created the following processing is performed:
 - a. If a matching instance data node is found, the user interface control will be connected to that element.
 - b. If a matching instance data node is not found, an instance data node will be created using the binding expression from the user interface control as the `name`. If the `name` is not a valid QName, processing halts with an exception ([4.5.1 The `xforms-binding-exception` Event](#)).

The above steps comprise the default processing of `xforms-model-construct-done`.

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

4.2.3 The `xforms-ready` Event

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

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info: None

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

4.2.4 The `xforms-model-destruct` Event

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

Target: `model`

Bubbles: No

Cancelable: No

Context Info: None

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

4.3 Interaction Events

4.3.1 The `xforms-rebuild` Event

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

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

All model item properties are initialized by processing all `bind` elements in document order. For each `bind`:

1. If the attribute `nodeset` is attached to the `bind`, it is evaluated to select an XPath node-set. Otherwise, if the `bind` does not have a `nodeset` attribute, then the selected XPath node-set consists of the in-scope evaluation context.
2. For each node in the selected XPath node-set, model item properties are applied according to the remaining attributes on the `bind` element (for details on the model item properties, see [6 Model Item Properties](#)). If a node already contains a model item property of the same name due to the processing of prior `bind` elements, then XForms processing for the containing document halts with an exception ([4.5.1 The xforms-binding-exception Event](#)).
3. For each node in the selected XPath node-set, any child `bind` elements are recursively processed as described in the three points of this list.

After initial processing of the `bind` elements, the computational dependency data structures are rebuilt, and then the change list *L* is set to contain references to all instance nodes that have an associated computational expression so that a *full* recalculation is performed the next time the behavior of `xforms-recalculate` is invoked.

4.3.2 The xforms-recalculate Event

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

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

The values of all instance data items match their associated 'calculate' constraints, if any. All model item properties that can contain computed expressions are resolved. In addition to contributing further node value changes that will cause `xforms-value-changed` notifications in `xforms-refresh`, the model item properties that change are marked to help `xforms-refresh` to determine the notification events to dispatch.

- If the required model item property changes, then either the `xforms-required` event must be marked for dispatch if `required` is true or the `xforms-optional` event must be marked for dispatch if `required` is false. Marking one of these events for dispatch unmarks the other.
- If the readonly model item property changes, then either the `xforms-readonly` event must be marked for dispatch if `readonly` is true or the `xforms-readwrite` event must be marked for dispatch if `readonly` is false. Marking one of these events for dispatch unmarks the other.
- If the relevant model item property changes, then either the `xforms-enabled` event must be marked for dispatch if `relevant` is true or the `xforms-disabled` event must be marked for dispatch if `relevant` is false. Marking one of these events for dispatch unmarks the other.

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

When it is time to recalculate a model item property, the XPath expression is evaluated. The evaluation context is determined from the model binding expression that applied the model item property, as defined for computed expressions in [7.2 Evaluation Context](#). The XPath expression may **reference** or **refer to** another instance node, in which case the value of the instance node is **referenced**. Each referenced instance node has as **dependents** those computes which directly refer to the instance node. References to the current node's value in `calculate` expressions are explicitly ignored, i.e., if an expression associated with a `compute` refers to the instance node associated with the `compute`, then the instance node does not take itself as a dependent. A `compute` is **computationally dependent** on an instance node (whose value may or may not be computed) if there is a path of dependents leading from the instance node through zero or more other instance nodes to the `compute`. A `compute` is part of a **circular dependency** if it is computationally dependent on itself.

Note:

Referring to a node's value in a `calculate` on the node, as in the following example, may have effects that vary by implementation: `<bind nodeset="x" calculate="..+1"/>`. Model item properties other than `calculate`, such as `required` or `readonly` are well-defined in the presence of self-references.

Note:

An example of a `calculate` formula that contains a self-reference (i.e. that refers to the node it calculates) appears in Section [6.1.2 The readonly Property](#). The example enforces a default value for a node and, as mentioned above, does not create a circular dependency. An example of a circular dependency is `<bind nodeset="A|B" calculate="../A + ../B"/>`. In this example, node *A* depends in part on *B*, and node *B* depends in part on *A*.

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

1. An XForms Processor must recalculate computes for nodes in *L*, if any, and nodes that are computationally dependent on nodes in *L*.
2. An XForms Processor must perform only a single recalculation of each compute that is computationally dependent on one or more of the elements in *L*.
3. An XForms Processor must recalculate a compute *C* after recalculating all computes of instance nodes on which *C* is computationally dependent. (Equivalently, an XForms Processor must recalculate a compute *C* before recalculating any compute that is computationally dependent on the instance node associated with *C*.)
4. Finally, if a compute is part of a circular dependency and also computationally dependent on an element in *L*, then an XForms processor must report an exception ([4.5.2 The xforms-compute-exception Event](#)).

[C Recalculation Sequence Algorithm](#) describes one possible method for achieving the required recalculation behavior.

4.3.3 The xforms-revalidate Event

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

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

An instance node is [valid](#) if and only if the following conditions hold:

- the constraint model item property is true
- the value is non-empty if the required model item property is true
- the node satisfies all applicable XML schema definitions (including those associated by the type model item property, by an external or an inline schema, or by `xsi:type`)

Note:

`xsi:type` attributes on instance data elements are processed even in the absence of external or inline schema.

Note:

The applicable XML schema definitions are determined as defined in Section [3.3.1 The model Element](#).

The default action for this event results in the following:

All instance data nodes in all `instance` elements in the `model` are checked for validity according to the above definition. If the validity of a node changes, then either the `xforms-valid` event must be marked for dispatch if the node changes from invalid to valid or the `xforms-invalid` event must be marked for dispatch if the node changes from valid to invalid. Marking one of these events for dispatch unmarks the other.

Note:

Since the event sequence for `xforms-model-construct` excludes `xforms-refresh` and discards event notification marks, form controls bound to invalid nodes do not receive an initial `xforms-invalid` event.

4.3.4 The xforms-refresh Event

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

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

1. All [UI Expressions](#) are reevaluated (implementations may optimize this operation but must behave as if all UI Expressions are reevaluated).
2. A node can be changed by a number of mechanisms in XForms, including confirmed user input to a form control, an `xforms-recalculate` ([4.3.2 The xforms-recalculate Event](#)), and the `setvalue` action ([10.2 The setvalue Element](#)). If the value of an instance data node was changed, then the node must be marked for dispatching the `xforms-value-changed` event.
3. If the `xforms-value-changed` event is marked for dispatching, then all of the appropriate model item property notification events must also be marked for dispatching (`xforms-optional` or `xforms-required`, `xforms-readwrite` or `xforms-readonly`, and `xforms-enabled` or `xforms-disabled`).
4. The user interface reflects the state of the model, which means that all forms controls and related UI elements reflect their corresponding instance data, including:
 - current values (for the appropriate form controls and related UI elements)
 - [validity](#)

- other model item properties (`required`, `readonly` and `relevant`).
- the proper number of and content for repeat objects.

This process includes sending the notification events to the form controls. For each form control, each notification event for which the form control is a legitimate target and that is marked for dispatching on the bound node must be dispatched (`xforms-value-changed`, `xforms-valid`, `xforms-invalid`, `xforms-optional`, `xforms-required`, `xforms-readwrite`, `xforms-readonly`, and `xforms-enabled`, `xforms-disabled`). The notification events `xforms-out-of-range` or `xforms-in-range` must also be dispatched as appropriate. This specification does not specify an ordering for the events.

4.3.5 The `xforms-reset` Event

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

Target: `model`

Bubbles: Yes

Cancelable: Yes

Context Info: None

The default action for this event results in the following:

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

4.3.6 The `xforms-next` and `xforms-previous` Events

Dispatched in response to: user request to navigate to the next or previous [Core Form Control](#).

Target: [Core Form Controls](#)

Bubbles: No

Cancelable: Yes

Context Info: None

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

Navigation is determined on a containing document-wide basis. The host language is responsible for defining overall navigation order. The following describes a possible technique based on a `navindex` attribute, using individual form controls as a navigation unit: The `<group>`, `<repeat>`, and `<switch>` structures serve as container navigation units that, instead of providing a single navigation point, create a local navigation context for child form controls (and possibly other substructures). The navigation sequence is determined as follows:

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

4.3.7 The `xforms-focus` Event

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

Target: [Core Form Control](#)`|group|switch|repeat`

Bubbles: No

Cancelable: Yes

Context Info: None

The default action for these events results in the following:

Focus is given to the target form control if the form control is able to accept focus. Changing the focus to a form control within a repeat object may cause one or more repeat index values to be changed as described in Section [9.3.4 User Interface Interaction](#). Setting focus to a `repeat` container form control sets the focus to the [repeat object](#) associated with the repeat index. Setting the focus to a `group` or `switch` container form control set the focus to the first form control in the container that is able to accept focus. Any form control is able to accept the focus if it is relevant.

Note:

This event is implicitly invoked to implement XForms accessibility features such as `accesskey` and when the user changes the focus.

4.3.8 The xforms-help and xforms-hint Events

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

Target: [Core Form Control](#)

Bubbles: Yes

Cancelable: Yes

Context Info: None

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

4.3.9 The xforms-submit Event

See chapter [11.2 The xforms-submit Event](#).

4.3.10 The xforms-submit-serialize Event

See chapter [11.3 The xforms-submit-serialize Event](#).

4.4 Notification Events**4.4.1 The xforms-insert Event**

Dispatched in response to: Successful insertion of one or more nodes by an XForms `insert` action.

Target: `instance`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
inserted-nodes	node-set	The instance data node or nodes inserted.
origin-nodes	node-set	The instance data nodes referenced by the <code>insert</code> action's <code>origin</code> attribute if present, or the empty nodeset if not present.
insert-location-node	node-set	The <code>insert location</code> node as defined by the <code>insert</code> action.
position	string	The insert position, <code>before</code> or <code>after</code> .

Default Action: None; notification event only.

Note:

Notification events are those with no default processing defined. Although this event is dispatched by `insert` processing as a notification, `repeat` processing associates behavior with the capture phase of this event.

4.4.2 The xforms-delete Event

Dispatched in response to: Successful deletion of one or more nodes by an XForms `delete` action.

Target: `instance`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
deleted-nodes	node-set	The instance data node or nodes deleted. Note that these nodes are no longer referenced by their parents.
delete-location	number	The <code>delete location</code> as defined by the <code>delete</code> action, or NaN if there is no delete location.

Default Action: None; notification event only.

Note:

Notification events are those with no default processing defined. Although this event is dispatched by `delete` processing as a notification, `repeat` processing associates behavior with the capture phase of this event.

4.4.3 The xforms-value-changed Event

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

Target: [Core Form Controls](#)

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event due to a change.

Note:

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

Note:

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

4.4.4 The xforms-valid Event

Dispatched in response to: an instance data node either changing and being or becoming [valid](#).

Target: [Core Form Controls](#)_{group|switch}

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.3 The xforms-revalidate Event](#).

4.4.5 The xforms-invalid Event

Dispatched in response to: an instance data node either changing and being or becoming invalid (not [valid](#)).

Target: [Core Form Controls](#)_{group|switch}

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.3 The xforms-revalidate Event](#).

4.4.6 The xforms-readonly Event

Dispatched in response to: an instance data node either changing and being or becoming readonly.

Target: [Core Form Controls](#)_{group|switch}

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.2 The xforms-recalculate Event](#) or [4.3.4 The xforms-refresh Event](#).

4.4.7 The xforms-readwrite Event

Dispatched in response to: an instance data node either changing and being or becoming read-write.

Target: [Core Form Controls](#)_{group|switch}

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.2 The xforms-recalculate Event](#) or [4.3.4 The xforms-refresh Event](#).

4.4.8 The xforms-required Event

Dispatched in response to: an instance data node either changing and being or becoming required.

Target: [Core Form Controls](#)_{group}switch

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.2 The xforms-recalculate Event](#) or [4.3.4 The xforms-refresh Event](#).

4.4.9 The xforms-optional Event

Dispatched in response to: an instance data node either changing and being or becoming optional.

Target: [Core Form Controls](#)_{group}switch

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.2 The xforms-recalculate Event](#) or [4.3.4 The xforms-refresh Event](#).

4.4.10 The xforms-enabled Event

Dispatched in response to: an instance data node either changing and being or becoming enabled.

Target: [Core Form Controls](#)_{group}switch

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.2 The xforms-recalculate Event](#) or [4.3.4 The xforms-refresh Event](#).

4.4.11 The xforms-disabled Event

Dispatched in response to: an instance data node either changing and being or becoming disabled.

Target: [Core Form Controls](#)_{group}switch

Bubbles: Yes

Cancelable: No

Context Info: None

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

This event is dispatched during [4.3.4 The xforms-refresh Event](#) if the bound instance data node has been marked for dispatching this event in [4.3.2 The xforms-recalculate Event](#) or [4.3.4 The xforms-refresh Event](#).

4.4.12 The DOMActivate Event

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

Target: [Core Form Controls](#)

Bubbles: Yes

Cancelable: Yes

Context Info: None

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

4.4.13 The DOMFocusIn Event

Dispatched in response to: a form control receiving focus.

Target: [Core Form Controls](#)|group|switch|repeat

Bubbles: Yes

Cancelable: No

Context Info: None

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

4.4.14 The DOMFocusOut Event

Dispatched in response to: a form control losing focus.

Target: [Core Form Controls](#)|group|switch|repeat

Bubbles: Yes

Cancelable: No

Context Info: None

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

4.4.15 The xforms-select and xforms-deselect Events

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

Target: `item` or `case`

Bubbles: Yes

Cancelable: No

Context Info: None

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

4.4.16 The xforms-in-range Event

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

Target: [Core Form Controls](#)

Bubbles: Yes

Cancelable: No

Context Info: None

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

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

4.4.17 The xforms-out-of-range Event

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

Target: [Core Form Controls](#)

Bubbles: Yes

Cancelable: No

Context Info: None

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

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

4.4.18 The xforms-scroll-first and xforms-scroll-last Events

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

Target: `repeat`

Bubbles: Yes

Cancelable: No

Context Info: None

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

4.4.19 The `xforms-submit-done` Event

See chapter [11.4 The `xforms-submit-done` Event](#).

4.5 Error Indications

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

4.5.1 The `xforms-binding-exception` Event

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

Target: any element that can contain a binding expression

Bubbles: Yes

Cancelable: No

Context Info: None

Default Action: Fatal error (halts processing).

4.5.2 The `xforms-compute-exception` Event

Dispatched as an indication of: an error occurring during XPath evaluation for a model item property (see [6 Model Item Properties](#)).

Target: `model`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
error-message	string	An implementation-specific string that should contain the expression being processed when the exception was detected.

Default Action: Fatal error (halts processing).

4.5.3 The `xforms-version-exception` Event

Dispatched as an indication of failure of the version checks defined in the description of the `version` attribute in Section [3.3.1 The `model` Element](#).

Target: the default `model`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
error-information	string	An implementation-specific error string

Default Action: Fatal error (halts processing).

Note:

This exception occurs early in processing. XForms processors are not expected to product XForms user interface elements nor even execute XForms action handlers (such as a `message` action) in response to this event. This exception is dispatched for the benefit of implementation-specific processing code that may be monitoring the behavior of an XForms processor.

4.5.4 The `xforms-link-exception` Event

Dispatched as an indication of: a failure to traverse or process the result of a link in a situation critical to form processing, such as schema or instance initialization.

Target: `model` associated with the in-scope evaluation context node of the element performing the link

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
resource-uri	string	The URI associated with the failed link (xsd:anyURI)

Default Action: Fatal error (halts processing).

Note:

This exception occurs early in processing. XForms processors are not expected to produce XForms user interface elements nor even execute XForms action handlers (such as a `message` action) in response to this event. This exception is dispatched for the benefit of implementation-specific processing code that may be monitoring the behavior of an XForms processor.

4.5.5 The `xforms-output-error` Event

Dispatched by the processor immediately after the failure of an `output` to render or update the rendition of content.

Target: `output`

Bubbles: Yes

Cancelable: No

Context Info: None.

Default Action: None; notification event only.

Note:

The `output` element content can include XForms actions, so an `output` element can contain an event handler for the `xforms-output-error` event. See Section [8.1.5 The output Element](#).

4.5.6 The `xforms-submit-error` Event

See chapter [11.5 The xforms-submit-error Event](#).

4.6 Event Sequencing

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

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

- When the form control is interactively changed, and has the `incremental="true"` setting, the event sequence described at [4.6.7 Sequence: Value Change](#) may be initiated at implementation dependent intervals.
- When the form control is interactively changed and does not have the `incremental="true"` setting, no events are required to be dispatched, and thus no order is defined.
- When the user activates the control and the value has changed, then, after the new value is placed into the bound instance node, the event sequence consists of the events described at [4.6.7 Sequence: Value Change](#) followed by dispatching the DOMActivate event. See [8.1.2 The input Element](#) for an example.
- When focus changes from the form control and the value has changed, then, after the new value is placed into the bound instance node, the event sequence is as described at [4.6.7 Sequence: Value Change](#).

4.6.2 For `output` Controls

- The implementation of an `output` dispatches `xforms-output-error` if it unable to process the output data (such as corrupt image data when the output mediatype indicates it is image data). This event may occur each time the `output` is given new data (such as a change of data in the bound data node or a change of the bound data node).

4.6.3 For `select` or `select1` Controls

- When a selection is interactively changed, and the form control has the `incremental="true"` setting (which is the default for the `select` or `select1` elements), the event sequence is described at [4.6.6 Sequence: Selection Without Value Change](#), which may be followed immediately by the sequence described at [4.6.7 Sequence: Value Change](#).
- When a selection is interactively changed, and the `select` or `select1` form control has the `incremental="false"` setting, the event sequence is described at [4.6.6 Sequence: Selection Without Value Change](#).
- When the user activates the control and the selection has changed, then, after the new value is placed into the bound instance node, the event sequence consists of the events described at [4.6.7 Sequence: Value Change](#) followed by dispatching the DOMActivate event. Note that this event sequence will have been preceded by the event sequence described at [4.6.6 Sequence: Selection Without Value Change](#) at the moment the selection was changed.

- When focus changes from the form control and the selection has changed, then, after the new value is placed into the bound instance node, the event sequence is as described at [4.6.7 Sequence: Value Change](#). Note that this event sequence will have been preceded by the event sequence described at [4.6.6 Sequence: Selection Without Value Change](#) at the moment the selection was changed.

4.6.4 For `trigger` Controls

- Activating the form control causes the event sequence defined at [4.6.8 Sequence: Activating a Trigger](#).

4.6.5 For `submit` Controls

- Activating the form control causes the event sequence defined at [4.6.8 Sequence: Activating a Trigger](#), followed immediately by the event sequence defined at [4.6.9 Sequence: Submission](#).

4.6.6 Sequence: Selection Without Value Change

1. `xforms-deselect` (for each `item` deselected by the change, if any)
2. `xforms-select` (for each `item` selected by the change, if any)

4.6.7 Sequence: Value Change

1. `xforms-recalculate`
2. `xforms-revalidate`
3. `xforms-refresh` performs reevaluation of UI binding expressions then dispatches these events according to value changes, model item property changes and validity changes:
 - [n] `xforms-value-changed`
 - [n] `xforms-valid` or `xforms-invalid`
 - [n] `xforms-enabled` or `xforms-disabled`
 - [n] `xforms-optional` or `xforms-required`
 - [n] `xforms-readonly` or `xforms-readwrite`
 - [n] `xforms-out-of-range` or `xforms-in-range`

(The order in which these events are dispatched is not defined).

4. Perform further deferred updates as necessary

4.6.8 Sequence: Activating a Trigger

1. `DOMActivate`

4.6.9 Sequence: Submission

1. `xforms-submit`
2. `xforms-submit-serialize`
3. `xforms-submit-done` or `xforms-submit-error`

4.7 Resolving ID References in XForms

The element of a document for which an IDREF must be resolved is called the **source element**, and the element bearing the matching ID, if there is one, is called the **target element**. Due to the run-time expansion of repeated content in XForms, it is possible that there will be more than one occurrence of both the source and target elements. This section describes how XForms IDREF resolution works to accommodate such repetition of the originating document's content.

Each run-time occurrence of the source element is called a **source object**, and each run-time occurrence of the target element is called a **target object**. It is the source object that performs the IDREF resolution, and the result of the search is either null or a target object.

Whether or not repeated content is involved, a null search result for an IDREF resolution is handled differently depending on the source object. If there is a null search result for the target object and the source object is an XForms action such as `dispatch`, `send`, `setfocus`, `setindex` or `toggle`, then the action is terminated with no effect. Similarly, a `submit` form control does not dispatch `xforms-submit` if its `submission` attribute does not indicate an existing `submission` element. Likewise, when an XPath function associated with the source object performs the IDREF search and a null result is obtained, the function returns an empty result such as `NaN` for the `index()` function or empty `nodeset` for the `instance()` function. However, an `xforms-binding-exception` occurs if there is a null search result for the target object indicated by attributes `bind`, `model` and `instance`.

If the target element is not repeated, then the search for the target object is trivial since there is only one associated with the target element that bears the matching ID. This is true regardless of whether or not the source object is repeated. However, if the target element is repeated, then additional information must be used to help select a target object from among those associated with the identified target element.

4.7.1 References to Elements within a repeat Element

When the target element that is identified by the IDREF of a source object has one or more `repeat` elements as ancestors, then the set of ancestor repeats are partitioned into two subsets, those in common with the source element and those that are not in common. Any ancestor

`repeat` elements of the target element not in common with the source element are descendants of the `repeat` elements that the source and target element have in common, if any.

For the `repeat` elements that are in common, the desired target object exists in the same set of run-time objects that contains the source object. Then, for each ancestor `repeat` of the target element that is not in common with the source element, the current index of the `repeat` determines the set of run-time objects that contains the desired target object.

4.7.2 References to Elements within a bind Element

When a source object expresses a Single Node Binding or Node Set Binding with a `bind` attribute, the IDREF of the `bind` attribute is resolved to a target bind object whose associated nodeset is used by the Single Node Binding or Node Set Binding. However, if the target `bind` element has one or more `bind` element ancestors, then the identified `bind` may be a target element that is associated with more than one target bind object.

If a target `bind` element is outermost, or if all of its ancestor `bind` elements have `nodeset` attributes that select only one node, then the target `bind` only has one associated bind object, so this is the desired target bind object whose nodeset is used in the Single Node Binding or Node Set Binding. Otherwise, the in-scope evaluation context node of the source object containing the `bind` attribute is used to help select the appropriate target bind object from among those associated with the target `bind` element.

From among the bind objects associated with the target `bind` element, if there exists a bind object created with the same in-scope evaluation context node as the source object, then that bind object is the desired target bind object. Otherwise, the IDREF resolution produced a null search result.

4.8 DOM Interface for Access to Instance Data

For each `model` element, the XForms Processor maintains the state in an internal structure called [instance data](#) that conforms to the XPath Data Model [\[XPath 1.0\]](#). XForms Processors that implement DOM must provide DOM access to this instance data via the interface defined below.

Note:

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

The IDL for this interface follows:

```
#include "dom.idl"

pragma prefix "w3c.org"

module xforms {
    interface XFormsModelElement : dom::Element {
        dom::Document getInstanceDocument(in dom::DOMString instanceID)
            raises(dom::DOMException);
        void rebuild();
        void recalculate();
        void revalidate();
        void refresh();
    };
};
```

4.8.1 The getInstanceDocument() Method

If the `instance-id` parameter is the empty string, then the document element of the default instance is returned. Otherwise, this method returns a DOM Document that corresponds to the instance data associated with the `instance` element containing an ID matching the `instance-id` parameter. If there is no matching instance data, a `DOMException` is thrown.

The implementation of the DOM interface for the instance document must not permit direct mutations of readonly instance nodes. Specifically, the implementation must not allow insertion of a node whose parent is readonly, direct deletion of a readonly node, nor setting the content of a readonly node. A node that is not readonly can be deleted, including all descendants, even if it has readonly descendants.

4.8.2 The rebuild() Method

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

4.8.3 The recalculate() Method

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

4.8.4 The revalidate() Method

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

4.8.5 The refresh() Method

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

4.9 Feature string for the hasFeature method call

For this version of the XForms specification, the feature string for the [\[DOM2 Core\]](#) `DOMImplementation` interface `hasFeature` method call is

"org.w3c.xforms.dom" and the version string is "1.0".

5 Datatypes

This chapter defines the datatypes used in defining an [XForms Model](#).

5.1 XML Schema Built-in Datatypes

XForms supports all XML Schema 1.0 [datatypes](#) except for `xsd:duration`, `xsd:ENTITY`, `xsd:ENTITIES`, and `xsd:NOTATION`. Concepts [value space](#), [lexical space](#) and constraining [facets](#) are as specified in [\[XML Schema part 2\]](#). XForms Processors must treat these datatypes as in-scope without requiring the inclusion of an XML Schema.

Note:

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

5.2 XForms Datatypes

XForms defines the following types in the XForms namespace. These datatypes can be used in the `type` model item property without a namespace prefix when the default namespace is the XForms namespace. All of these datatypes allow empty content. XForms Processors must treat these datatypes as in-scope without requiring the inclusion of an XML Schema.

5.2.1 Additional XForms Datatypes to Allow Empty Content

Many default XML schema types report empty content as invalid, which conflicts with the use of the `required` model item property. The following XForms datatypes are defined as having a lexical space consisting of either the empty string or the lexical space of the corresponding XML schema datatype. Although some XML schema datatypes do allow empty string content, they have also been added to the available XForms datatypes for form authoring consistency.

Built-in primitive types (in the XForms namespace):

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

Built-in derived types (in the XForms namespace):

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

5.2.2 xforms:listItem

This datatype serves as a base for the [listItems](#) datatype. The lexical space for `listItem` permits one or more characters valid for `xsd:string`, except white space characters.

5.2.3 xforms:listItems

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

Note:

In most cases, it is better to use markup to distinguish items in a list. See [9.3.6 The itemset Element](#).

5.2.4 xforms:dayTimeDuration

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

The `dayTimeDuration` datatype is made available by the XForms processor based on the following lexical space definition:

xforms:dayTimeDuration type definition

```
<xs:simpleType name="dayTimeDuration">
  <xs:restriction base="xsd:string">

    <xs:pattern value="([\\-]?P([0-9]+D(T([0-9]+(H([0-9]+(M([0-9]+(\\.([0-9]*)?S
      |\\.([0-9]+S)?|\\.([0-9]*)?S)|\\.([0-9]*)?S)?|M([0-9]+
      (\\.([0-9]*)?S|\\.([0-9]+S)?|\\.([0-9]*)?S)|\\.([0-9]+S)))?
      |T([0-9]+(H([0-9]+(M([0-9]+(\\.([0-9]*)?S|\\.([0-9]+S)?
      |\\.([0-9]*)?S)|\\.([0-9]*)?S)?|M([0-9]+(\\.([0-9]*)?S|\\.([0-9]+S)?
      |\\.([0-9]*)?S)|\\.([0-9]+S)))?)?"/>

  </xs:restriction>
</xs:simpleType>
```

5.2.5 xforms:yearMonthDuration

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

The `yearMonthDuration` datatype is made available by the XForms processor based on the following lexical space definition:

xforms:yearMonthDuration type definition

```
<xs:simpleType name="yearMonthDuration">
  <xs:restriction base="xsd:string">

    <xs:pattern value="([\\-]?P[0-9]+(Y([0-9]+M)?|M))?" />

  </xs:restriction>
</xs:simpleType>
```

5.2.6 xforms:email

This datatype represents an email address, as defined by [RFC 2822](#). Internationalized email addresses are not restricted by XForms beyond the definition in the RFC. For simplicity, some extremely uncommon features of the RFC syntax are not allowed, such as "Obsolete Addressing" from section 4.4, square-bracketed "domain-literal"s, and insignificant whitespace and comments.

Examples of valid `xforms:email` addresses

```
editors@example.com
~my_mail+{nosпам}$?@sub-domain.example.info
```

Examples of invalid `xforms:email` addresses

```
editors@(this is a comment)example.info
editors{at}example{dot}info
mailto:editors@example.com
```

Note:

The string `mailto:editors@example.com` is a valid `xsd:anyURI` but not a valid `xforms:email` because the colon separator for the URI scheme is not allowed before the '@' symbol. A valid `xforms:email` address does not include a `mailto:` URI scheme.

Note:

It is outside the scope of XForms to determine whether a given email address actually corresponds to an active mailbox.

The email datatype is made available by the XForms processor based on the following lexical space definition:

xforms:email type definition

```
<xs:simpleType name="email">
  <xs:restriction base="xsd:string">

    <xs:pattern value="([A-Za-z0-9!#-'\*\+\-\/=?\^_`{|}~]+(\.[A-Za-z0-9!#-'\*\+\-\/=?\^_`{|}~]+)*@[A-Za-z0-9!#-'\*\+\-\/=?\^_`{|}~]+(\.[A-Za-z0-9!#-'\*\+\-\/=?\^_`{|}~]+)*)" />

  </xs:restriction>
</xs:simpleType>
```

5.2.7 xforms:card-number

This type defines the basic lexical properties of a datatype that can be used to represent various ID, debit and credit card numbers.. The lexical space of the `xforms:card-number` datatype is a pattern restriction on `xsd:string`: it must be zero or more digits (0 - 9).

Note:

The display representation of this datatype by form controls is not required to match the lexical space of the bound instance data. User agents should apply appropriate conventions to the display and input of values, including separator characters.

xforms:card-number type definition

```
<xs:simpleType name="card-number">
  <xs:annotation>
    <xs:documentation>
      This type defines the basic lexical properties for a datatype that can be used to represent
      various ID numbers such as for debit and credit cards.
      This type does not apply the Luhn checksum algorithm.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xsd:string">
    <xs:pattern value="[0-9]*/>
  </xs:restriction>
</xs:simpleType>
```

The standard defines the structure of the number as well as how to apply the Luhn formula to ensure a correct check digit. This type only specifies the format of the number. The complementary XPath function `is-card-number()` should be used to validate that the ID number conforms to the specification.

Credit Card Example

```
<model xmlns="http://www.w3.org/2002/xforms">
  <instance>
    <payment method="cc" xmlns="http://commerce.example.com/payment">
      <number>4111111111111111</number>
      <expiry/>
    </payment>
  </instance>
  <bind nodeset="number" type="card-number" constraint="is-card-number(.)"/>
</model>
```

This example specifies that the element `number` is of the type `card-number` and that to be valid the `is-card-number()` function must evaluate to true indicating that check digit is valid.

6 Model Item Properties

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

6.1 Model Item Property Definitions

Model item properties can be distinguished along various axes.

Computed expressions vs. fixed properties:

- Fixed properties are static values that the XForms Processor evaluates only once. Such properties consist of literals, and are not subject to XPath evaluation.
- [Computed expressions](#) are XPath expressions that provide a value to the XForms Processor. Such values are recalculated at certain times as specified by the XForms Processing Model (see [4 Processing Model](#)). These expressions encode dynamic properties, often constraints, such as the dependency among various data items. Computed expressions are not restricted to examining the value of the instance data node to which they apply. XPath expressions provide the means to traverse the instance data; more complex computations may be encoded as call-outs to external scripts.

Inheritance rules:

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

Note:

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

Assigning local values:

Local values are assigned by processing all `bind` elements in an XForms Model in document order. It is an error to attempt to set a model item property twice on the same node (see [4.3.1 The xforms-rebuild Event](#) for details)..

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

Description
Computed Expression (yes or no)

Legal Values
Default Value
Inheritance Rules

6.1.1 The type Property

Description: The `type` model item property can be applied to both elements and attributes. The `type` model item property is not applied to instance nodes that contain child elements. The `type` model item property associates a datatype (as defined in [XML Schema part 2](#)) with the string-value (as defined in [XPath 1.0](#)) of an instance node. The datatype being associated can be obtained from a `simpleType` definition or a `simpleContent` definition from a `complexType`. If the datatype cannot be obtained as just described, then the Default Value of `xsd:string` is used. This model item property does not prevent form controls and XForms actions from setting invalid values into data nodes.

Computed Expression: No.

Legal Values: Any `xsd:QName` representing a datatype definition in an XML Schema. The namespace context from the parent `bind` of the `type` attribute is used to resolve the namespace qualification of the value.

Default Value: `xsd:string`.

Inheritance Rules: does not inherit.

This model item property contributes to the overall validity assessment of a node; the effect of validity state on bound form controls is described in [Section 8.1.1 Implementation Requirements Common to All Form Controls](#).

Note:

In XML Schema, an element can be made nillable. Although this means it can have empty content, nillable is defined by [XML Schema part 1](#) to be a property of an element, not a type. Therefore, the nillable property from XML Schema cannot be applied to instance nodes using the `type` model item property.

Associating datatypes with instance nodes

```
<model xmlns:my="http://example.org">

  <xs:schema targetNamespace="http://example.org" xmlns:my="http://example.org">
    <xs:simpleType name="Currency">
      <xs:restriction base="xsd:string">
        <xs:enumeration value="USD"/>
        <xs:enumeration value="EUR"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="Price">
      <xs:simpleContent>
        <xs:extension base="xsd:double">
          <xs:attribute name="currency" type="my:Currency" use="optional" default="USD"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:schema>

  <instance>
    <data xmlns="http://example.org">
      <aString>Hello, world!</aString>
      <simpleType>
        <price>100.00</price>
        <price>abc</price>
        <price currency="EUR">100.00</price>
        <price currency="EUR">abc</price>
      </simpleType>
      <complexType>
        <price>100.00</price>
        <price>abc</price>
        <price currency="abc">100.00</price>
        <price currency="EUR">abc</price>
      </complexType>
    </data>
  </instance>

  <bind nodeset="my:aString" type="xsd:string"/>
  <bind nodeset="my:simpleType/my:price" type="xsd:double"/>
  <bind nodeset="my:complexType/my:price" type="my:Price"/>
  <bind nodeset="my:complexType/my:price[3]/@currency" type="my:Currency"/>
  <bind nodeset="/my:data" type="xsd:string"/>

</model>
```

The first bind expresses the default datatype of `xsd:string`.

The second and third binds place type model item properties on each of the four `price` element children of the elements `simpleType` and `complexType`. Both binds associate the datatype `xsd:double` with the nodes. In both cases, the first and third nodes are considered valid according to the type model item property because their content matches the `xsd:double` datatype constraint. For both binds, the second and fourth `price` nodes are not valid due to their content.

The fourth bind places a type model item property on the `currency` attribute of the third `price` element. According to this association, the `currency` attribute node is not valid because its content does not match the enumeration given for `my:Currency`. Note that the containing element `price` is valid according to its type model item property.

The fifth bind attempts to associate a datatype with the `data` element. The association is ignored since the `data` element contains child elements.

6.1.2 The readonly Property

Description: describes whether the node content is restricted from changing.

Computed Expression: Yes.

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

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

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

Note:

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

When `true`, this model item property indicates that the XForms Processor must not allow any direct changes to the content of the bound instance data node from constructs other than the model item property system (i.e. other than a `calculate`). Instance mutations performed by `submission`, form controls, DOM interface access, and XForms actions must not insert or copy a new node into a parent node that is `readonly`, delete or replace a node whose parent is `readonly`, nor change the value or content of a `readonly` node. A node that is `readonly` but whose parent is not `readonly` can be entirely deleted or replaced by a submission even though doing so indirectly results in deletion or replacement of `readonly` descendant nodes.

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

Attaching a readonly property

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

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

The following example illustrates the ability to override the default `readonly` setting on calculated nodes.

Setting a default value

```
<model>
  <instance>
    <my:data></my:data>
  </instance>
  <bind nodeset="/my:data" readonly="false()" calculate="choose(.='', 'default', .)"/>
</model>
<input ref="/my:data"> ...
```

The `calculate` on `my:data` is executed on any `recalculate` that follows a rebuild, including the form initialization, so the user initially sees the word 'default'. The user may make any change to `my:data` with the `input`, and the calculation will be executed again as a result. Therefore, if the user enters an empty value, then the `calculate` will change `my:data` back to 'default'.

6.1.3 The required Property

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

Computed Expression: Yes.

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

Default Value: `false()`.

Inheritance Rules: does not inherit.

A form may *require* certain values, and this requirement may be dynamic. When evaluating to `true`, this model item property indicates that a non-empty instance data node is required before a submission of instance data can occur. Non-empty is defined as: The value of the bound instance data node must be convertible to an XPath `string` with a length greater than zero.

Note:

The XML Schema feature represented by `nilable` and `xsi:nil` is unrelated to the XForms `required` model item property. An element may have the `xsi:nil` attribute set to `true` to indicate that its empty content is schema valid, but if the `required` model item property for that element node is `true`, then the element violates the `required` constraint because a required node must be non-empty as defined above.

Except as noted below, the `required` model item property does not provide a hint to the XForms user interface regarding visibility, focus, or navigation order. XForms authors are strongly encouraged to make sure that form controls that accept `required` data are visible. An XForms Processor must provide an indication that a form control is required, and may provide immediate feedback, including limiting navigation. This model item property does not prevent form controls and XForms actions from setting empty strings into data nodes.

Attaching a required property

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

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

Note:

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

6.1.4 The relevant Property

Description: indicates whether the model item is currently *relevant*. Instance data nodes with this property evaluating to `false` are unavailable in the user interface and can be removed from submission serialization.

Computed Expression: Yes.

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

Default Value: `true()`.

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

Note:

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

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

Through single node UI bindings, the `relevant` model item property provides information to the XForms user interface regarding visibility, focus, and navigation order. In general, when `true`, associated form controls should be made available for user interaction. When `false`, associated form controls (and any children) and group and switch elements (including content) must be made unavailable, removed from the navigation order, and not allowed focus. Typically, non-relevant user interface content is not presented, or it may be styled as disabled. Elements other than form controls may also use a single node binding that selects a non-relevant node, but such elements are not made unavailable or non-operable due to the single node binding because it is not a [UI binding expression](#). For example, actions such as [10.2 The setvalue Element](#) and [10.16 The message Element](#) or the [11.1 The submission Element](#) remain operable if their single node bindings select a non-relevant node. However, some such elements may indirectly be affected by the `relevant` model item property. For example, it is possible for non-relevant nodes to be excluded from the data of a submission. Similarly, non-relevance indirectly affects the running of actions because a non-relevant form control disables event handlers that listen for events targeted at the form control element.

Note:

A core form control, group or switch must express a single node binding in order to be associated with an instance node. Due to the definition of [repeat object](#), the `relevant` model item property of the node in the associated [repeat item](#) affects the availability of the repeat object.

Attaching a relevant property

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

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

6.1.5 The calculate Property

Description: supplies an expression used to calculate a string value for the associated instance data node.

Computed Expression: Yes.

Legal Values: Any XPath expression.

Default Value: none.

Inheritance Rules: does not inherit.

An XForms Model may include model items whose string values are computed from other values. For example, the sum over line items for

quantity times unit price, or the amount of tax to be paid on an order. The formula for such a computed value can be expressed with a `calculate` property, whose XPath expression is evaluated, converted to a string with the XPath `string()` function, and stored as the value content of the calculated data node. Chapter [4 Processing Model](#) contains details of when and how the calculation is performed.

Attaching a calculate property

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

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

6.1.6 The constraint Property

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

Computed Expression: Yes.

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

Default Value: `true()`.

Inheritance Rules: does not inherit.

When evaluating to XPath `false`, the associated model item is not valid; the converse is not necessarily true. This model item property does not prevent form controls and XForms actions from setting invalid values into data nodes. Chapter [4 Processing Model](#) contains details of when and how the constraint is calculated as well as when validation is performed. This model item property contributes to the overall validity assessment of a node; the effect of validity state on bound form controls is described in Section [8.1.1 Implementation Requirements Common to All Form Controls](#).

Attaching a constraint property

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

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

Note:

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

6.1.7 The p3ptype Property

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

Computed Expression: No.

Legal Values: `xsd:string`.

Default Value: none

Inheritance Rules: does not inherit.

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

Attaching a type constraint using Binding

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

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

6.2 Schema Constraints

Chapter [5 Datatypes](#) described how XForms uses the XML Schema datatype system to constrain the [value space](#) of data values collected by an XForms Model. Such datatype constraints can be provided via an XML Schema. Alternatively, this section lists various mechanisms for attaching type constraints to instance data. Attributes `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` are ignored for purposes for locating a Schema.

6.2.1 Atomic Datatype

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

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

1. Applicable XML schema definitions (including those associated an external or an inline schema, or by `xsi:type`)
2. An XForms `type` constraint associated with the instance data node using [XForms binding](#).
3. If no type constraint is provided, the instance data node defaults to `type="xsd:string"` (default to string rule).

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

<p>Type Constraint Using XML Schema</p> <pre><xs:simpleType name="nonEmptyString"> <xs:restriction base="xsd:string"> <xs:minLength value="1"/> </xs:restriction> </xs:simpleType></pre>
<p>This new datatype would then be associated with one or more model items through one of the methods outlined here.</p>
<p>Attaching A Type Constraint</p> <pre><my:first-name xsi:type="my:nonEmptyString"/></pre>
<p>This defines element <code>first-name</code> to be of type <code>my:nonEmptyString</code>.</p>
<p>Attaching Type Constraint Using XForms Binding</p> <pre><instance> <my:first-name /> </instance> <bind type="my:nonEmptyString" nodeset="/my:first-name"/></pre>
<p>Here, we have attached type information to element <code>first-name</code> via element <code>bind</code>. Thus the XForms author can extend external schemas without having the ability to change them.</p>

7 XPath Expressions in XForms

XForms uses XPath to address [instance data nodes](#) in binding expressions, to express constraints, and to specify calculations. XPath expressions in XForms are based on [XPath 1.0](#). A future version of XForms is expected to enable use of [XPath 2.0](#). At the time of evaluation, an XPath expression must be syntactically correct. In addition, the namespaces the expression references must be in scope and the functions and variables it references must be defined. If any of these conditions is not satisfied, an exception ([4.5.2 The xforms-compute-exception Event](#)) is raised, except for binding expressions, which produce a different exception ([4.5.1 The xforms-binding-exception Event](#)).

7.1 XPath Datatypes

XPath datatypes are used only in [binding expressions](#) and [computed expressions](#). XForms uses XPath datatypes `boolean`, `string`, `number`, and `node-set`. Additionally, the type `object` is used to denote a parameter or return value that can be any one of the four XPath datatypes. A future version of XForms is expected to use XPath 2.0, which includes support for XML Schema datatypes.

7.2 Evaluation Context

Within XForms, the **default model** is the first `model` in document order. The **default instance** of any `model` is the first child `instance` in document order within that `model`. XPath expressions appearing in various XForms attributes are used to reference instance data. Every XPath expression requires an evaluation context consisting of a node, position, size, variable bindings, function set, and namespace context. For all evaluation contexts in XForms,

1. No variable bindings are in place.
2. The available function library is defined below in [7.5 The XForms Function Library](#).
3. Any namespace declarations in scope for the attribute that defines the expression are applied to the expression.
4. The context node, position and size are determined according to rules described below.

A **binding element** is any element that is explicitly allowed to have a binding expression attribute, and a **bound element** is any element that explicitly declares a binding expression attribute. A binding expression attribute contains an XPath expression that references zero or more nodes of instance data. Every XPath expression requires an evaluation context. The **in-scope evaluation context** of a binding element provides an evaluation context for the binding expression attribute. The following rules are used in determining the node, position and size of the in-scope evaluation context:

1. A binding element is "**outermost**" if the element has no ancestor binding elements. If an outermost binding element is contained by a `model`, then the context node for the outermost binding element is the top-level document element node of the default instance of the containing `model` element. Otherwise, the context node for outermost binding elements is the top-level document element node of the default instance in the default model. For outermost binding elements, the context size and position are 1.
2. The context node, position and size for non-outermost binding elements is determined using the binding expression attribute or in-scope evaluation context of the nearest ancestor binding element. This is also referred to as "scoped resolution". For a non-outermost binding element:
 - a. If the nearest ancestor binding element is not a bound element, then the in-scope evaluation context of the non-outermost binding element is equivalent to the in-scope evaluation context of the nearest ancestor binding element.
 - b. If the nearest ancestor binding element expresses a Single-Node binding, then the in-scope evaluation context of the non-outermost binding element has a context size and position of 1 and the context node is the one resulting from the Single-Node binding of the nearest ancestor binding element.
 - c. If the nearest ancestor binding element expresses a Node Set binding, then the XForms processor dynamically generates an occurrence of the non-outermost binding element for each of the nodes in the Node Set Binding of the nearest ancestor binding element. The **dynamic in-scope evaluation context** for each generated occurrence of the non-outermost binding element has a context size equal to the size of the nodeset identified by the Node Set Binding of the nearest ancestor binding element, the context node is the node for which the occurrence of the non-outermost binding element was generated, and the position is equal to the position of the generator node in the nodeset identified by the Node Set Binding of the nearest ancestor binding element.
 - d. If the nearest ancestor binding element expresses a Node Set binding, then the non-outermost binding element has its own in-scope evaluation context separately from those of its dynamically generated occurrences described above. The in-scope evaluation context has a context position of 1, and the context node and size are set by the first node and the size of the nodeset identified by the Node Set binding of the nearest ancestor binding element.
3. Once the context node of the in-scope evaluation context has been determined according to the rules above, if the binding element expresses a `model` attribute that refers to a `model` other than the one containing the context node, then the context node of the in-scope evaluation context is changed to be the top-level document element node of the default instance of the referenced `model`, and the context position and size are changed to 1.

The **in-scope evaluation context** of an element that is not a binding element is the same as if the element were a binding element. For example, the in-scope evaluation context for the `setindex` action element is required to provide the context for evaluating the `index` attribute, so it is determined as if the element could contain a binding expression attribute.

XPath expressions also appear in model item property attributes of the `bind` element to define computed expressions. If the `bind` element does not express a Node Set binding, then the in-scope evaluation context for model item property attributes of the `bind` is equal to the in-scope evaluation context for the `bind`. Otherwise, the `bind` has a Node Set binding, and computed expressions for each model item property attribute are generated for each node. For each computed expression generated, the evaluation context node, position and size are determined by the same method as dynamic in-scope evaluation context rule above, except the computed expression is used in lieu of a binding expression attribute such that the `bind` element is the nearest ancestor binding element.

XPath expressions also appear in the special attributes of several other XForms elements, such as the `value` attribute on `setvalue` and `output`, the `at` attribute of `insert` and `delete`, or the `index` attribute of `setindex`. Generally, if the containing element does not express a Single Node Binding or Node Set Binding, then the special attribute is evaluated using the in-scope evaluation context. Special attributes may be evaluated using the in-scope evaluation context even if the containing element expresses a Single Node Binding or Node Set Binding. However, for some special attributes, the evaluation context node, position and size are based on the result of the Single Node Binding or Node set Binding. Each special attribute that contains an XPath expression describes how its evaluation context node, position and size are determined.

In the following example, the `group` has a binding expression of `level2/level3`. According to the rules above, this outermost element node would have a context node of `/level1`, which is the document element node of the instance data. The `select1` form control then inherits a context node from the parent `group`.

Sample XML Instance Data

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

Binding Expression Context Nodes

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

This section describes how the in-scope evaluation context of an element is determined, not whether the in-scope evaluation will be used. The Single-Node Binding or Node Set Binding of a non-outermost binding element is not evaluated if the in-scope evaluation context does not contain a context node. This can occur if the Single-Node Binding or Node Set Binding of the nearest ancestor bound element produces an empty nodeset. Also, if the Single-Node Binding or Node Set Binding of an element is expressed with the `bind` attribute, then the resulting node or nodeset is obtained from the referenced `bind` element. The `nodeset` attribute of a `bind` element is evaluated using the in-scope evaluation of the `bind` element, not the in-scope evaluation context of an element that references it with a `bind` attribute.

7.3 References, Dependencies, and Dynamic Dependencies

An XPath expression **references** a node of instance data if the node is selected during the evaluation of the expression. A node is selected by matching an XPath [NodeTest](#) or by being returned by a function call. For examples, a node can match a name test, a wildcard test, a node type test, or it can be returned by or used as a parameter to a function, or it can appear in a filter expression (where all of the prior examples recursively apply). Once selected, a node is considered to be referenced even if a filter expression subsequently excludes the node from further participation in the expression evaluation. The **reference list** of an XPath expression is the set of instance nodes that it references.

Example of References and Non-References

Given the following default instance:

```
<xforms:instance>
  <data xmlns="">
    <a attr="X">
      <b attr="Y">
        <c/>
      </b>
    </a>
    <a attr="Z">
      <b attr="Z">
        <c/>
      </b>
    </a>
  </data>
</xforms:instance>
```

and the following XPath expression:

```
a[@attr='X']/b[@attr='X']/c
```

Both nodes named `a` are referenced since both are matched by a `NameTest`. The `attr` attribute in each element `a` is referenced during the evaluation of the filter expression. The filter expression rejects the second element `a`, but that element is still considered to have been referenced because it was selected for further processing during the expression evaluation.

The element named `b` in the first element `a` is referenced, but element `b` in the second `a` is not referenced because the expression evaluation did not proceed beyond the filter expression that rejected the second `a` element.

While performing the `NameTest` for element `b`, observe that an XPath expression evaluator may visit all the children of the first element `a` in order to perform the `NodeTest`. However, a node is not referenced if it is only visited but fails the `NodeTest`. In this case, the `NodeTest` is a `NameTest` for `b`, which the element `d` fails. Therefore, `d` is not referenced.

The filter expression test on `b` rejects the only element `b` that has been selected so far because the attribute value of `attr` does not match the equality test. Still, `b` and its attribute `attr` have been referenced by this expression.

Element `c` is not considered to be referenced by this expression given this data. Although a `NameTest` for `c` appears in the expression, the evaluation of the expression did not proceed to perform the `NameTest` due to the rejection of `b` by the filter expression.

Finally, note that an XPath expression can reference many nodes even if its final result is an empty nodeset.

Note:

Defining a **reference** in terms of matching a `NodeTest` was a deliberate design decision that creates more references than necessary in order to make the computation system more responsive to certain types of changes without needing a rebuild operation. When a leaf node is filtered from an expression by a predicate, the leaf node is still considered to be referenced so that if the condition changes such that the leaf node would be included in the result value of the expression, then the expression will be recalculated without needing a rebuild. However, once a node is rejected from an expression, further location steps are not evaluated relative to the rejected node, so references for that location step are only created based on its execution relative to accepted nodes. For example, if the above expression were in a `calculate`, and if the attribute of either `b` element changes to the value `x`, the expression would be recalculated but it still does not record a reference to any `c` elements until the references are obtained in the next rebuild operation. Moreover, the excess references created by the definition may cause some recalculation constructs to cease operation due to circular references that would not be created with a stricter definition of a reference. A future version of XForms may use a stricter version of referencing for recalculation and a less strict definition of referencing for the purpose of detecting and performing automatic rebuild operations.

The referencing of a repeat index by the `index()` function is handled as a special case. Implementations must behave as if each occurrence of a `repeat` is associated with an implicitly managed instance data node containing the repeat index. If a `repeat` identified as `R1` contains a `repeat` identified as `R2`, then a repeat index instance node is managed for each occurrence of `R2` that is generated by `R1`. An invocation of the function `index()` in an XPath expression is considered to reference the repeat index instance node corresponding to the repeat index value that it returns.

If an XPath expression references an instance node, then the expression result is **dependent** on the instance node. A **dependency list** for an instance node is the list of XPath expressions of a given category that are dependent upon the instance node. For example, in Section [4.3.2 The xforms-recalculate Event](#), the dependency list of computed expressions for each instance node helps establish the recalculation order for the computed expressions when the values of instance data nodes are changed.

The references of an XPath expression may be altered by insertion of instance nodes since the new nodes may be referenced by the XPath expression if it is re-evaluated. Similarly, the references of an XPath expression may be altered by deletion of instance nodes that are being referenced by the XPath expression. An XPath expression is **dynamically dependent** on an instance node if its reference list is altered by inserting, deleting or changing the value of the instance node. An instance node is a **dynamic dependency** for an XPath expression if the expression is dynamically dependent on the instance node. If an XPath expression contains a dynamic dependency and the XForms processor is maintaining dependency lists for the category of the XPath expression, then changing the dynamic dependency implies a change to the dependency lists of instance nodes referenced by the XPath expression.

The computational dependency data structure described in Section [4.3.2 The xforms-recalculate Event](#) essentially stores the dependency lists of instance nodes corresponding to all the references made by computed expressions (see Appendix [C Recalculation Sequence Algorithm](#) for details). The computational dependency data structure is not reconstructed in response to a dynamic dependency change. Instead, the form author may request a rebuild of the computational dependency data structures using the `rebuild` action. Additionally, the `insert` and `delete` actions set a rebuild flag so that computational dependency data structures will be rebuilt at the end of an action sequence.

Due to the rebuild flag setting on `insert` and `delete`, a form author can use many kinds of dynamic dependencies in model binding expressions

and computed expressions without ever explicitly invoking the `rebuild` action. This includes the use of functions such as `position()`, `last()`, and `count()` on element and attribute nodes because the return values of the functions in these cases is fixed except when an `insert` or `delete` occurs. By comparison, functions such as `id()`, `instance()`, and `index()` can establish dynamic dependencies that can necessitate invoking a `rebuild` if they are used in model binding expressions or computed expressions because the results of the functions are affected by changing the values of instance nodes, not by inserting or deleting nodes.

7.4 Expression Categories

There are several different categories of XPath expressions used in XForms, and they are processed at different times depending on the category. A [binding expression](#) is an XPath expression used to bind a [model item property](#) to one or more instance nodes, or to bind a form control to instance data, or to specify the node or node set for operation by an action. The evaluation schedule for binding expressions differs based on whether the binding expression is a model binding expression, UI binding expression, or a binding expression for an XForms action. XPath expressions are also used in various other attributes of XForms binds, actions, form controls, and `submissions`, and their descendant elements. These expressions follow one of the three schedules associated with binding expressions as described below.

7.4.1 Model Binding Expressions and Computed Expressions

A [model binding expression](#) is a kind of binding expression that can be used to declare model item properties, and is used in the Node-Set binding of the `bind` element. A [computed expression](#) is an XPath expression used to determine the value of a [model item property](#) based on instance data. Several of the attributes of `bind` are computed expressions, including `calculate`, `readonly`, `relevant`, and `required`.

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

Note:

If the `index()` function is being invoked from a model binding expression or computed expression, it will be necessary to invoke `rebuild` manually. If the repeat index change occurs due to an implicit behavior such as a change to the focused form control, then the `rebuild` (along with `recalculate`, `revalidate` and `refresh`) can be invoked from a handler for `DOMFocusIn` attached to the `repeat` or each repeat object.

7.4.2 Expressions in Actions and Submissions

Binding expressions on XForms actions and XPath expressions appearing in other attributes of XForms actions are evaluated at the time the XForms action is performed. In some cases, XForms actions have child elements that allow the values of some of their attributes to be determined based on instance data. In these cases, the `value` attribute of the child element is evaluated at the time the corresponding attribute is needed in the processing model of the containing XForms action.

Similarly, XPath expressions used in the attributes of `submission` and its child elements (other than XForms actions) are evaluated as needed within the submission processing model.

Form authors can use dynamic dependencies in the XPath expressions of XForms Actions and Submissions without invoking any special data structure reconstruction actions because implementations must behave as if these expressions are evaluated as needed.

7.4.3 UI Expressions

A [UI Binding Expression](#) is a Single Node Binding or Node Set Binding in a form control. A **UI expression** is a UI Binding Expression or an XPath expression appearing in a descendant element of a form control, except for XPath expressions in the categories described above (such as XForms action expressions). These include the Single Node Bindings and Node Set Bindings of the additional elements that contribute to the behavior of a form control (including `label`, `help`, `hint` and `alert`, `filename` and `mediatype`) and the selection helper elements (`itemset`, `label`, `value` and `copy`). This also includes the `value` attribute on `output` and `value` elements.

Form authors can use dynamic dependencies in UI Expressions without invoking any special data structure reconstruction actions because the state of the user interface at the end of processing [xforms-refresh](#) is required to reflect the instance data as if all UI Expressions had been re-evaluated.

Note:

Implementations may record UI Expression dependency lists on instance nodes to help streamline detection of the need to re-evaluate a UI Expression at the beginning of processing for [xforms-refresh](#). Such implementations may determine that a UI Expression is stale (needs re-evaluation) when a node on which it depends has been changed. An implementation may also indicate that all UI Expressions are stale if a node of instance data is inserted, deleted or replaced, or the implementation may streamline detection of which UI Expressions may be affected by the insertion, deletion or replacement of an instance data node.

7.4.4 UI Binding in other XML vocabularies

The XForms binding mechanism allows other XML vocabularies to make single node bindings between custom user interface controls and XForms instance data. As an example, XForms binding attribute `bind` might be used within XHTML 1.x user interface controls as shown below. See [3.2.3 Single-Node Binding Attributes](#).

XForms Binding In XHTML 1.x User Interface Controls

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

7.4.5 Binding Examples

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

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

```

    </orderForm>
  </xforms:instance>
  <xforms:bind nodeset="/orderForm/shipTo/firstName" id="fn" />
</xforms:model>

```

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

UI Binding Using Attribute `ref`

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

UI Binding Using Attribute `bind`

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

Specifies Model Containing The Instance Explicitly

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

7.5 The XForms Function Library

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

The following sections define additional required functions for use within XForms : [7.6 Boolean Functions](#), [7.7 Number Functions](#), [7.8 String Functions](#), [7.9 Date and Time Functions](#), [7.10 Node-set Functions](#) , and [7.11 Object Functions](#).

The function library provided by an XForms processor may also contain other extension functions as described in [7.12 Extension Functions](#).

If an error occurs in an XPath function, then an [4.5.2 The xforms-compute-exception Event](#) or [4.5.1 The xforms-binding-exception Event](#) occurs..

7.6 Boolean Functions

7.6.1 The `boolean-from-string()` Function

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

Function `boolean-from-string` returns `true` if the required parameter `string` is "true" or "1", or `false` if parameter `string` is "false", or "0". This is useful when referencing a Schema `xsd:boolean` datatype in an XPath expression. If the parameter `string` matches none of the above strings, according to a case-insensitive comparison, the return value is `false`.

7.6.2 The `is-card-number()` Function

boolean **is-card-number**(*string?*)

If the string parameter conforms to the pattern restriction of the `card-number` datatype, then this function applies the Luhn algorithm described in [\[Luhn Patent\]](#) and returns `true` if the number satisfies the formula. Otherwise, `false` is returned. If the parameter is omitted, it defaults to the string-value of the current context node.

Examples (see also [5.2.7 xforms:card-number](#)):

```
is-card-number(.)
```

returns `true` if and only if the context node contains a string that contains of zero or more digits and satisfies the formula.

```
is-card-number('4111111111111111')
```

returns `true`. Other examples of string constants that will return `true` are : 5431111111111111, 3411111111111111 and 6011601160116611.

```
is-card-number('123')
```

returns `false`.

7.7 Number Functions

7.7.1 The `avg()` Function

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

Function `avg` returns the arithmetic average of the result of converting the string-values of each node in the argument node-set to a number. The sum is computed with `sum()`, and divided with `div` by the value computed with `count()`. If the parameter is an empty node-set, or if any of the nodes evaluate to `NaN`, the return value is `NaN`.

7.7.2 The `min()` Function

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

Function `min` returns the minimum value of the result of converting the string-values of each node in argument `node-set` to a number. "Minimum"

is determined with the < operator. If the parameter is an empty node-set, or if any of the nodes evaluate to NaN, the return value is NaN.

7.7.3 The max() Function

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

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

7.7.4 The count-non-empty() Function

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

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

7.7.5 The index() Function

number **index**(*string*)

Function **index** takes a string argument that is the IDREF of a *repeat* and returns the current 1-based position of the repeat index for the identified *repeat*—see [9.3.1 The repeat Element](#) for details on *repeat* and its associated repeat index. If the specified argument does not identify a *repeat*, the function returns NaN.

Note:

The IDREF obtained from the function parameter may not uniquely identify the desired *repeat* if the *repeat* element bearing the matching ID resides in a repeating construct such as element *repeat*. The general method described in [4.7 Resolving ID References in XForms](#) is used to determine the desired run-time repeat object.

index

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

7.7.6 The power() Function

number **power**(*number*, *number*)

Raises the first argument to the power of the second argument, returning the result. If the calculation does not result in a real number, then NaN is returned.

Examples:

```
power(2, 3)
```

returns 8

```
power(-1, 0.5)
```

returns NaN.

```
if (prin>0 and dur>0 and rate>0, prin*rate/(1-power(1+rate, -dur)), 0)
```

returns a compounded interest payment value given a non-zero principal (*prin*), duration (*dur*) and periodic interest rate (*rate*).

7.7.7 The random() Function

number **random**(*boolean?*)

This function generates and returns a uniformly distributed random or pseudorandom number in the range from 0.0 up to but excluding 1.0. This function accepts an author-optional boolean parameter that is *false* by default. If *true*, the random number generator for this function is first seeded with a source of randomness before generating the return value. A typical implementation may seed the random number generator with the current system time in milliseconds when **random**(*true*) is invoked, and it may apply a linear congruential formula to generate return values on successive invocations of the function.

Example:

```
random()
```

could return 0.14159265358979

7.7.8 The compare() Function

number **compare**(*string*, *string*)

This function returns -1, 0, or 1, depending on whether the value of the first argument is respectively less than, equal to, or greater than the value of second argument based on lexicographic comparison using Unicode code point values [\[Unicode Collation Algorithm\]](#).

Example:

```
compare('apple', 'orange')

returns -1
```

7.8 String Functions

7.8.1 The if() Function

string if(boolean, string, string)

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

Note:

This function is deprecated because a future version of XForms is expected to be based on [XPath 2.0](#), which contains an `if` construct that is incompatible with this function. Form authors and design tools are encouraged to use the function `choose()` from Section [7.11.1 The choose\(\) Function](#) instead of this function.

7.8.2 The property() Function

string property(string)

This function accepts a string identifying a property name. If the property name is not recognized, empty string is returned. The property definitions for this function are as follows:

Property	Return Value
version	1.1
conformance-level	full, basic Or a string beginning with full or basic
Any other NCName	Reserved. Their use results in an exception (see 7.5 The XForms Function Library for the exception type)
QNameButNotNCName	An implementation-specific property value, such as a locale or timezone for the user agent. If the implementation does not support the property, then empty string is returned.

Examples:

```
property('version')

returns 1.1

property('conformance-level')

may return full
```

7.8.3 The digest() Function

string digest(string, string, string?)

This function accepts a string of data, a string indicating a cryptographic hashing algorithm, and an author-optional string indicating an encoding method. The data string is serialized as UTF-8, the hash value is then computed using the indicated hash algorithm, and the hash value is then encoded by the indicated method, and the result is returned by the function. The following table presents the keywords for the second string parameter and the corresponding hash algorithms:

Keywords	Hash Algorithm	Status
MD5	The MD5 hash algorithm defined in [MD5]	Required
SHA-1	The SHA-1 hash algorithm defined in [SHA2]	Required
SHA-256	The SHA-256 hash algorithm defined in [SHA2]	Required
SHA-384	The SHA-384 hash algorithm defined in [SHA2]	Optional
SHA-512	The SHA-512 hash algorithm defined in [SHA2]	Optional
Any other NCName	Reserved. Their use results in an exception (see 7.5 The XForms Function Library for the exception type)	Required
QNameButNotNCName	An implementation-specific hash algorithm is used. If the implementation does not support the indicated hash algorithm, then an exception occurs (see 7.5 The XForms Function Library for the exception type).	Required

This recommendation defines the values `hex` and `base64` for the third string parameter that indicates the encoding method. If the parameter is missing, then the default is `base64`. The `hex` and `base64` encoding methods of this function correspond to the encodings defined in [XML Schema part 2](#) for the datatypes `hexBinary` and `base64Binary`, respectively. For the hexadecimal encoding, the digits 'a' through 'f' are encoded with lower case letters. Any other string value given for the encoding method results in an exception (see [7.5 The XForms Function Library](#) for the exception type).

<code>digest('abc', 'SHA-1', 'hex')</code>
returns a9993e364706816aba3e25717850c26c9cd0d89d.
<code>digest('abc', 'MD5', 'hex')</code>
returns 900150983cd24fb0d6963f7d28e17f72.
<code>digest('abc', 'SHA-256', 'hex')</code>
returns ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

7.8.4 The hmac() Function

string **hmac**(*string*, *string*, *string*, *string*?)

This function accepts a string for a key or shared secret, a string of data, a string indicating a cryptographic hashing algorithm, and an author-optional string indicating an encoding method. The key and data strings are serialized as UTF-8, and they are subjected to the HMAC algorithm defined in [\[HMAC\]](#) and parameterized by the the hash algorithm indicated by the third parameter. The result is encoded with the method indicated by the fourth parameter, and the result is returned by the function.

The following table presents the keywords for the third string parameter and the corresponding hash algorithms:

Keywords	Hash Algorithm	Status
MD5	The MD5 hash algorithm defined in [MD5]	Required
SHA-1	The SHA-1 hash algorithm defined in [SHA2]	Required
SHA-256	The SHA-256 hash algorithm defined in [SHA2]	Required
SHA-384	The SHA-384 hash algorithm defined in [SHA2]	Optional
SHA-512	The SHA-512 hash algorithm defined in [SHA2]	Optional
Any other NCName	Reserved. Their use results in an exception (see 7.5 The XForms Function Library for the exception type)	Required
QNameButNotNCName	An implementation-specific hash algorithm is used. If the implementation does not support the indicated hash algorithm, then an exception occurs (see 7.5 The XForms Function Library for the exception type).	Required

This recommendation defines the values `hex` and `base64` for the fourth string parameter that indicates the encoding method. If the parameter is missing, then the default is `base64`. The `hex` and `base64` encoding methods of this function correspond to the encodings defined in [XML Schema part 2](#) for the datatypes `hexBinary` and `base64Binary`, respectively. For the hexadecimal encoding, the digits 'a' through 'f' are encoded with lower case letters. Any other string value given for the encoding method results in an exception (see [7.5 The XForms Function Library](#) for the exception type).

<code>hmac('Jefe', 'what do ya want for nothing?', 'SHA-1', 'hex')</code>
returns effcdf6ae5eb2fa2d27416d5f184df9c259a7c79
<code>hmac('Jefe', 'what do ya want for nothing?', 'MD5', 'hex')</code>
returns 750c783e6ab0b503eaa86e310a5db738
<code>hmac('Jefe', 'what do ya want for nothing?', 'SHA-256', 'hex')</code>
returns 5bdcc146bf60754e6a042426089575c75a003f089d2739839dec58b964ec3843

7.9 Date and Time Functions

Note:

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

7.9.1 The local-date() Function

string **local-date**()

This function returns a lexical `xsd:date` obtained as if by the following rules: the result of `now()` is converted to a local date based on the user agent time zone information. If no time zone information is available, then the date portion of the result of `now()` is returned.

Example:

<code>local-date()</code>
could return 2006-10-13-07:00

```
substring(local-date(), 1, 10)
```

could return 2006-10-13

```
days-to-date(days-from-date(local-date()) + 31)
```

returns a date that is 31 days from today. For example, if `local-date()` returns 2006-10-13-07:00, then the result is 2006-11-13.

7.9.2 The `local-dateTime()` Function

string `local-dateTime()`

This function returns a lexical `xsd:dateTime` obtained as if by the following rules: the result of `now()` is converted to a local `dateTime` based on the user agent time zone information. If no time zone information is available, then the result of `now()` is returned.

Example:

```
local-dateTime()
```

could return 2006-10-13T16:04:17-07:00

```
adjust-dateTime-to-timezone(seconds-to-dateTime(seconds-from-dateTime(local-dateTime()) + 7200))
```

Adds two hours (7200 seconds) to the local date and time, returning the result in the local time zone. For example, if `local-dateTime()` returns 2007-10-02T14:26:43-07:00, then the above expression returns 2007-10-02T16:26:43-07:00

7.9.3 The `now()` Function

string `now()`

The `now` function returns the current UTC date and time as a string value in the canonical XML Schema `xsd:dateTime` format. If time zone information is available, it is used to convert the date and time to UTC. If no time zone information is available, then the date and time are assumed to be in UTC.

Note:

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

```
now()
```

returns 2006-10-14T01:04:17Z if `local-dateTime()` returns 2006-10-13T18:04:17-07:00

```
seconds-to-dateTime(seconds-from-dateTime(now()) + 7200)
```

Computes two hours from now, returning the result in UTC time. For example, if `now()` returns 2007-10-02T21:26:43Z, then the above expression returns 2007-10-02T23:26:43Z

7.9.4 The `days-from-date()` Function

number `days-from-date(string)`

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

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

Note:

If an `xsd:date` is given as the parameter, the timezone is ignored if provided because there is no way to normalize to the date in the UTC timezone without both the time and timezone.

Examples:

```
days-from-date("2002-01-01")
```

returns 11688

```
days-from-date("2002-01-01-07:00")
```

returns 11688

```
days-from-date("1969-12-31")
```

returns -1

7.9.5 The days-to-date() Function

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

This function returns a string containing a lexical `xsd:date` that corresponds to the number of days passed as the parameter according to the following rules:

The number parameter is rounded to the nearest whole number, and the result is interpreted as the difference between the desired date and 1970-01-01. An input parameter value of NaN results in output of the empty string.

Examples:

<code>days-to-date(11688)</code>
returns 2002-01-01
<code>days-to-date(-1)</code>
returns 1969-12-31

7.9.6 The seconds-from-dateTime() Function

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

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

If the string parameter represents a legal lexical `xsd:dateTime`, the return value is equal to the number of seconds difference between the specified `dateTime` (normalized to UTC) and 1970-01-01T00:00:00Z. If no time zone is specified, UTC is used. Any other input string parameter causes a return value of NaN. This function does not support leap seconds.

Example:

<code>seconds-from-dateTime('1970-01-01T00:00:00Z')</code>
returns 0
<code>seconds-from-dateTime('1970-01-01T00:00:00-08:00')</code>
returns 28800

7.9.7 The seconds-to-dateTime() Function

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

This function returns a string containing a lexical `xsd:dateTime` that corresponds to the number of seconds passed as the parameter according to the following rules:

The number parameter is rounded to the nearest whole number, and the result is interpreted as the difference between the desired UTC `dateTime` and 1970-01-01T00:00:00Z. An input parameter value of NaN results in output of the empty string. This function does not support leap seconds.

Examples:

<code>seconds-to-dateTime(0)</code>
returns 1970-01-01T00:00:00Z
<code>seconds-from-dateTime(28800)</code>
returns 1970-01-01T08:00:00Z
<code>seconds-to-dateTime(seconds-from-dateTime(now()) + 7200)</code>
Computes two hours from now, returning the result in UTC time. For example, if <code>now()</code> returns 2007-10-02T21:26:43Z, then the above expression returns 2007-10-02T23:26:43Z
<code>adjust-dateTime-to-timezone(seconds-to-dateTime(seconds-from-dateTime(now()) + 7200))</code>
Computes two hours from now, returning the result in the local time zone. For example, if <code>now()</code> returns 2007-10-02T21:26:43Z and the local date and time is 2007-10-02T14:26:43-07:00, then the above expression returns 2007-10-02T16:26:43-07:00

7.9.8 The adjust-dateTime-to-timezone() Function

string **adjust-dateTime-to-timezone**(*string*)

This function adjusts a legal lexical `xsd:dateTime` received as the string parameter to the local time zone of the implementation, and returns the result. If the string argument contains no time zone, then the result is the string argument with the local time zone as the time zone component. If the implementation does not have access to time zone information, UTC is used. The result is empty string if the string argument is the empty sequence or not a legal lexical `xsd:dateTime`.

Examples:

<code>adjust-dateTime-to-timezone('2007-10-07T02:22:00')</code>
returns 2007-10-07T02:22:00-07:00 in the Pacific time zone since daylight savings time applies.
<code>adjust-dateTime-to-timezone('2007-10-02T21:26:43Z')</code>
returns 2007-10-02T14:26:43-07:00 in the Pacific time zone since daylight savings time applies.
<code>adjust-dateTime-to-timezone(seconds-to-dateTime(seconds-from-dateTime(now()) + 7200))</code>
Computes two hours from now, returning the result in the local time zone. For example, if <code>now()</code> returns 2007-10-02T21:26:43Z and the local date and time is 2007-10-02T14:26:43-07:00, then the above expression returns 2007-10-02T16:26:43-07:00

7.9.9 The seconds() Function

number seconds(string)

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

If the string parameter represents a legal lexical `xsd:duration`, the return value is equal to the number specified in the seconds component plus 60 * the number specified in the minutes component, plus 60 * 60 * the number specified in the hours component, plus 60 * 60 * 24 * the number specified in the days component. The sign of the result will match the sign of the duration. Year and month components, if present, are ignored. Any other input parameter causes a return value of NaN.

Note:

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

Examples:

<code>seconds("P3DT10H30M1.5S")</code>
returns 297001.5 (3 days, 10 hours, 30 minutes, and 1.5 seconds)
<code>seconds("P1Y2M")</code>
returns 0 because the year and month parts of the duration are ignored and the remaining portions are unspecified and default to 0
<code>seconds("3")</code>
returns NaN because the parameter is not a lexically valid duration

7.9.10 The months() Function

number months(string)

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

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

Note:

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

Examples:

Examples:

<code>months("P1Y2M")</code>
returns 14 (1 year and 2 months)
<code>months("-P19M")</code>
returns -19 because the duration is negative and expresses 0 years and 19 months

7.10 Node-set Functions

7.10.1 The instance() Function

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

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

If the argument is omitted or is equal to the empty string, then the root element node (also called the document element node) is returned for the default instance in the model that contains the current context node.

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

Example:

For instance data corresponding to this XML:

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

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

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

7.10.2 The current() Function

node-set **current**()

Returns the [context node](#) used to initialize the evaluation of the containing XPath expression.

Examples:

For the following instance data:

```
<xforms:instance xmlns="">
  <converter>
    <amount>100</amount>
    <currency>jpy</currency>
    <convertedAmount></convertedAmount>
  </converter>
</xforms:instance>

<xforms:instance xmlns="" id="convTable">
  <convTable date="20040212" currency="cdn">
    <rate currency="eur">0.59376</rate>
    <rate currency="mxn">8.37597</rate>
    <rate currency="jpy">80.23451</rate>
    <rate currency="usd">0.76138</rate>
  </convTable>
</xforms:instance>
```

and the following value calculation bind:

```
<bind nodeset="convertedAmount"
  calculate="../amount *
    instance('convTable')/rate[@currency=current()../currency]"/>
```

the content value of `/converter/convertedAmount` is the product of `/converter/amount` and the conversion table rate given by the `rate` element whose `currency` attribute value matches the content of `/converter/currency`.

For the following instance data:

```
<xforms:instance xmlns="" id="i1">
  <months>
    <mon>01</mon>
    <mon>02</mon>
    <mon>03</mon>
  </months>
</xforms:instance>
<xforms:instance xmlns="" id="i2">
  <months>
    <month code="01">Jan</month>
    <month code="02">Feb</month>
```

```
<month code="03">Mar</month>
</months>
</xforms:instance>
```

and the following repeat structure:

```
<repeat nodeset="mon">
  <output value="instance('i2')/month[@code = current()]/>
</repeat>
```

the output should contain Jan Feb Mar.

7.10.3 The id() Function

node-set id(object, node-set?)

The `object` parameter provides one or more IDREFs. This may be in the form of a string containing a space-separated list of IDREFs or a node-set, each node of which contains an IDREF. The `node-set` parameter provides nodes in one or more instance data documents to be searched. If the `node-set` parameter is not given or is empty, then the instance data document to be searched is the one containing the context node of the function call. For each node in the `node-set` parameter (or its default), the set of element nodes are collected with IDs that match the IDREFs from the `object` parameter. The result of this function is a node-set containing the union of the collected element nodes from each string. An element node can be assigned an ID by means of an `xml:id` attribute or an attribute that is assigned the type ID by a DTD or `xsd:ID` or any type derived from `xsd:ID` by an XML schema, or the `type` model item property.

Example:

```
id('X Y', instance('Z'))
```

Returns nodes identified by X or Y from the instance data document associated with the root element of the `instance` identified by Z.

7.10.4 The context() Function

node-set context()

This function returns the in-scope evaluation context node of the nearest ancestor element of the node containing the XPath expression that invokes this function. The nearest ancestor element may have been created dynamically as part of the run-time expansion of repeated content as described in Section [4.7 Resolving ID References in XForms](#).

Example:

```
<setvalue ref="x" value="context()/y"/>
```

This action sets node `x` to the value of node `y`, where both nodes are children of the in-scope evaluation context node for the `setvalue` element.

Note:

An intended use of this function is in conjunction with the `repeat` element (Section [9.3.1 The repeat Element](#)) and the `setvalue` action element (Section [10.2 The setvalue Element](#)). The intent is to provide form authors with a means of expressing a `value` attribute that is relative to the repeat context node when the Single Node Binding result is not.

7.11 Object Functions

7.11.1 The choose() Function

object choose(boolean, object, object)

This function provides a conditional test that chooses an object to return based on the boolean parameter. If the boolean parameter is true, then the first object is returned, otherwise the second object is returned. Each of the object parameters can be of any XPath datatype as described in Section [7.1 XPath Datatypes](#), and this function does no type conversion of the parameter it chooses to return.

Note:

All parameters of an XPath function are evaluated, so the parameter that is not returned by this function is still evaluated, and its result is discarded by this function.

Note:

Form authors and design tools are encouraged to use this function instead of the function `if()` described in Section [7.8.1 The if\(\) Function](#), which has been deprecated. Because this function returns an object instead of a string, migrating from `if()` to `choose()` may occasionally require conversion of the return result using the `string()` function.

Example:

```
choose(count(x) > 0, x, y)
```

Returns the node-set of matching `x` if it is non-empty and the node-set matching `y` otherwise.

```
choose (@x, @x, 0)
```

If the context node of the function contains attribute `x`, then the nodeset containing that attribute is returned. Otherwise, the number `0` is returned.

7.11.2 The event() Function

object **event**(*string*)

Function `event` returns context specific information determined by the *string* argument. The returned context information is an XPath object whose type and content depends upon the requested property. Each event describes what properties can be accessed by this function and the type and value that will be returned as the result.

The event context properties available for each event are provided in the sections that describe the events.

This function is intended for use in the XPath expressions of XForms actions. If invoked for any other XPath expression, such as a binding expression or model item property expression, this function returns the empty string. If this function is invoked from an XPath expression for an XForms action, then event context information is used from the most recently dispatched event whose action handler contains the XForms action.

Some properties defined for an event may be unavailable if certain prerequisite conditions were not met prior to the event being dispatched. Implementations may also add custom properties. If the event context information does not contain the property indicated by the *string* argument, then an empty node-set is returned.

Examples:

```
event('inserted-nodes')
```

If called from an `xforms-insert` event handler, a nodeset is returned containing the instance data node or nodes inserted.

7.12 Extension Functions

XForms documents may use additional XPath extension functions beyond those described here. A number of useful community extensions are defined at [\[EXSLT\]](#). The names of any such extension functions must be declared in attribute `functions` on element `model`. Such declarations are used by the XForms Processor to check against available extension functions. XForms Processors perform this check at the time the document is loaded, and halt processing by signaling an exception ([4.5.2 The xforms-compute-exception Event](#)) if the XForms document declares an extension function for which the processor does not have an implementation.

Note:

Explicitly declaring extension functions enables XForms Processors to detect the use of unimplemented extension functions at document load-time, rather than throwing a fatal exception ([4.5.1 The xforms-binding-exception Event](#) or [4.5.2 The xforms-compute-exception Event](#)) during user interaction. Failure by authors to declare extension functions will result in an XForms Processor potentially halting processing during user interaction with a fatal error.

8 Core Form Controls

This chapter covers the XForms view layer features for directly interacting with instance data and properties from the XForms Model. This includes features that provide data from the model to the view layer as well as features of the view layer that commit data collected from the user to the model.

8.1 The XForms Core Form Controls Module

[Form controls](#) are declared using markup elements, and their behavior refined via markup attributes. The core form controls are described in this module, including their attributes and their content models (their metadata elements). A **core form control** is an element that acts as a direct point of user interaction and often provides read, write, or read/write access to a node of instance data. See Section [9 Container Form Controls](#) for a description of [container form controls](#).

Core Form Control Element	Attributes	Minimal Content Model
input	Common , UI Common , Single Node Binding , <code>inputmode (xsd:string)</code> , <code>incremental (xsd:boolean)</code>	label, (UI Common)*
secret	Common , UI Common , Single Node Binding , <code>inputmode (xsd:string)</code> , <code>incremental (xsd:boolean)</code>	label, (UI Common)*
textarea	Common , UI Common , Single Node Binding , <code>inputmode (xsd:string)</code> , <code>incremental (xsd:boolean)</code>	label, (UI Common)*
output	Common , Single Node Binding (author-optional), <code>appearance ("full" "compact" "minimal" QNameButNotNCName)</code> , <code>value (XPath Expression)</code> , <code>mediatype (xsd:string)</code>	label?, <code>mediatype?</code> , (UI Common)*
upload	Common , UI Common , Single Node Binding , <code>mediatype (xsd:string)</code> , <code>incremental (xsd:boolean)</code>	label, <code>filename?</code> , <code>mediatype?</code> , (UI Common)*
range	Common , UI Common , Single Node Binding , <code>start (xsd:string)</code> , <code>end (xsd:string)</code> , <code>step (xsd:string)</code> , <code>incremental (xsd:boolean)</code>	label, (UI Common)*
trigger	Common , UI Common , Single Node Binding (author-optional)	label, (UI Common)*
submit	Common , UI Common , Single Node Binding (author-optional), <code>submission (xsd:IDREF)</code>	label, (UI Common)*
select	Common , UI Common , Single Node Binding , <code>selection ("open" "closed")</code> , <code>incremental (xsd:boolean)</code>	label, (List UI Common)+, (UI Common)*

select1	Common , UI Common , Single Node Binding , selection ("open" "closed"), incremental (xsd:boolean)	label, (List UI Common)*, (UI Common)*
-------------------------	---	--

Note:

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

The following table summarizes additional support elements for form controls.

Support Element	Attributes	Minimal Content Model
label	Common , Single Node Binding (author-optional)	(PCDATA (UI Content))*
help	Common , Single Node Binding (author-optional)	(PCDATA (UI Content))*
hint	Common , Single Node Binding (author-optional)	(PCDATA (UI Content))*
alert	Common , Single Node Binding (author-optional)	(PCDATA (UI Content))*
filename (for upload)	Common , Single Node Binding	EMPTY
mediatype (for upload)	Common , Single Node Binding	EMPTY
mediatype (for output)	Common , Single Node Binding , value (string XPath Expression)	EMPTY
choices (for selection controls)	Common	label?, (List UI Common)+
item (for selection controls)	Common	label, value, (UI Common)*
value (for selection controls)	Common , Single Node Binding (author-optional)	PCDATA

See also: [9.3.6 The itemset Element](#) (for selection controls) and [9.3.7 The copy Element](#) (for selection controls).

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

Element	Attributes
(various)	appearance ("full" "compact" "minimal" QNameButNotNCName)

appearance

Author-optional attribute to define an appearance hint. If absent, the user agent may freely choose any suitable rendering.

Note:

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

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

navindex

This author-optional attribute is a non-negative integer in the range of 0-32767 used to define the navigation sequence. This gives the author control over the sequence in which [form controls](#) are traversed. The default navigation order is specified in the chapter [4 Processing Model](#).

accesskey

This author-optional attribute defines a shortcut for moving the input focus directly to a particular [form control](#). The value of this is a single character which when pressed together with a platform specific modifier key (e.g., the `alt` key) results in the focus being set to this [form control](#).

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

Additionally, this module defines the following content sets:

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

As shown above, the XML Events module adds the Actions content set into the UI Common content set. A host language may add markup to the [UI Content](#) set. When the XForms Extension module is present, it too should be included in the UI Common content set.

8.1.1 Implementation Requirements Common to All Form Controls

XForms user interface controls are bound to the underlying instance data using [binding](#) attributes as defined in the chapter [6 Model Item Properties](#).

Form controls enable accessibility by taking a uniform approach to such features as labels, help text, navigation, and keyboard shortcuts.

Internationalization issues are addressed by following the same design principles as in XHTML. All form controls are suitable for styling as aural or visual media.

Form controls encapsulate high-level semantics without sacrificing the ability to deliver real implementations. For instance, the form control `select` enables the user to *select items from a set*. These form controls distinguish the functional aspects of the underlying control from the presentational and behavioral aspects. This separation enables the expression of the intent underlying a particular form control—see [\[AUI97\]](#) for a definition of such high-level user interaction primitives.

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

All form controls must meet the following implementation requirements:

- All form controls, including [container form controls](#), should have an inline layout by default (e.g. for a host language that supports [CSS](#), the default styling should be `display:inline`). By default, [repeat items](#) should have a block layout (e.g. a default styling of `display:block` for host languages that support CSS).
- If a form control violates its data binding restriction, an `xforms-binding-exception` must occur.

Note:

Form controls that read or write `simpleContent` produce this exception whenever and as soon as they are bound to an element node that has an element child node.

- Form controls that write `simpleContent` to instance data must do so exactly as defined by the XForms Action `setValue` ([10.2 The setValue Element](#)).

Note:

If a form control binds to an element node, then regardless of how many child nodes the element has, the result of the form control writing to the bound element node is that it has either a single non-empty text node child, or no children if the `simpleContent` written is the empty string (which is in accord with the data model of [XPath 1.0](#)).

- All form controls that read `simpleContent` instance data must do so as follows:
 - Element nodes: If element child nodes are present, then an `xforms-binding-exception` occurs. Otherwise, return the string value of the node.
 - Attribute nodes: returns the string-value of the node.
 - Text nodes: returns the string-value of the node.
 - Namespace, processing instruction, and comment nodes: behavior is undefined (implementation-dependent).
 - the XPath root node: an `xforms-binding-exception` occurs.
- Form controls are considered to be **relevant** if none of the following apply and **non-relevant** if any of the following apply:
 - the Single Node Binding is expressed and resolves to empty nodeset,
 - the Single Node Binding is expressed and resolves to a non-relevant instance node,
 - the form control is contained by a non-relevant `switch` or `group` (which includes a non-relevant `repeat` item), or
 - the form control is contained by a non-selected `case` element of a `switch`.

When a form control becomes non-relevant, it must receive event `xforms-disabled` and then the XForms action handlers that are listening for events on the non-relevant form control must be disabled.

When a non-relevant form control changes to being relevant, the XForms action handlers that listen for events on the form control must become enabled and then the form control must be updated to represent the current value(s) and model item properties of the instance node(s) to which it is bound or to which it refers. The following events must be dispatched to the form control: `xforms-enabled`, `xforms-value-changed`, `one of xforms-valid` or `xforms-invalid`, `one of xforms-readonly` or `xforms-readwrite`, `one of xforms-required` or `xforms-optional`, and `one of xforms-in-range` or `xforms-out-of-range`.

- Except as noted, relevant form controls must distinguish rendering between being bound to a required node versus a non-required node. Exceptions are form controls that do not directly render the string value of the bound node (including `trigger` and the [container form controls](#)). Control of this behavior should be made available to stylesheets.
- Relevant form controls must distinguish rendering between valid and invalid states. Control of this behavior should be made available to stylesheets.
- Relevant form controls must indicate when the bound instance data contains a value or content that the form control is not capable of rendering. Control of this behavior should be made available to stylesheets.
- If a form control binds to a readonly node, then the form control must not allow the user to modify the node value. The relevant form control that is bound to a readonly node should render in a way which indicates that entering or changing the value is not allowed. Control of the render behavior should be made available to stylesheets.

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

Description
Common Attributes
Special Attributes

Examples
Data Binding Restrictions
Implementation Requirements

8.1.2 The input Element

Description: This form control enables free-form data entry or a user interface component appropriate to the datatype of the bound node..

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

inputmode

Author-optional. This form control accepts an input mode hint. [E Input Modes](#).

incremental

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

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

Note:

This control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

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

Examples:

Simple Free-Form Data Entry

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

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

A graphical browser might render the above example as follows:

Street

Datatype-sensitive Data Entry

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

A graphical browser might render the above example as follows:

Ship By

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

Ship By

April, 2002						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

Data Entry with Activation Behavior

```
<input ref="/search/expr">
  <label>Search term(s):</label>
  <send ev:event="DOMActivate" submission="doSearch" />
</input>

<submit submission="doSearch">
  <label>Search</label>
</submit>
```

This example shows the `input` and `submit` form controls working together to provide the common user experience of a simple search. The `input` collects the search term(s) and then automatically initiates the `submission` identified by `doSearch` if the user activates the `input` control. Alternatively, the user initiate the `doSearch` `submission` by activating the `submit` control.

Note:

A graphical browser on a desktop device might activate form controls such as `input` in response to a special user action such as pressing the Enter key or either clicking or double-clicking with the mouse (depending on the type of form control).

8.1.3 The secret Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

inputmode

Author-optional. This form control accepts an input mode hint. [E Input Modes](#).

incremental

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

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

Note:

This control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

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

Example:

Password Entry

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

A graphical browser might render this form control as follows:

Password

The password you enter will not be displayed

8.1.4 The textarea Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

inputmode

Author-optional. This form control accepts an input mode hint. [E Input Modes](#).

incremental

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

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

Note:

This control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

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

Example:

Email Message Body

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

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

A graphical browser might render the above example as follows:

Message Body:



8.1.5 The output Element

Description: This form control renders content based in part on instance data, but it provides no means for entering or changing data.

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

Special Attributes:

appearance

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

value

Author-optional. An XPath expression to be evaluated. The string result of the evaluation is rendered by the form control. If binding attributes are present to select a node, this attribute has no effect. The evaluation context is the same as would be applied to the evaluation of the single node binding. This XPath expression is re-evaluated whenever there is a change in any node to which the expression refers. An empty string is used if the XPath evaluation fails.

mediatype

Author-optional attribute used to indicate that data obtained from the Single-Node Binding should be rendered (after decoding, if needed) according to a desired media type indicated by the attribute value string, such as `image/*` for image rendition. If the `mediatype` element appears as a child of the `output`, then it overrides this attribute. If the media type is not specified by this attribute or by the `mediatype` element, then the default is to present the indicated data as plain text (with no decoding according to datatype).

Data Binding Restrictions: Binds to any simpleContent.

Note:

This control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

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

Element `output` can be used to display the value of a particular instance node by using a Single-Node Binding; it can also be used to display the result of evaluating an XPath expression by specifying the XPath expression to be evaluated via attribute `value`. Note that the Single-Node Binding attributes and `value` on element `output` are mutually exclusive.

By default, the `output` element simply renders the plain text of the `value` attribute or the node indicated by the Single-Node Binding. However, if the Single-Node Binding indicates a non-empty data node, and the media type is specified based on the [mediatype attribute](#) or [mediatype child element](#), then the content of the data node **must** be decoded or dereferenced according to its datatype, and the result **should** be rendered according to the indicated media type if it is possible to do so (e.g. a voice-only device cannot render a digital image).

Note:

When the media type is specified, implementations **may** handle the output content as presentation-only or as interactive content, and interactive content **may** be isolated from or capable of accessing the enclosing document that contains the `output`. Further implementation experience and user feedback is required. For example, if the output content includes XForms user interface elements, it may be desirable for them to access a default XForms model in the output content or from the enclosing document.

If the Single Node Binding is absent or if it does not indicate a non-empty instance node, then the media type specification is ignored if given. Otherwise, if the Single Node Binding produces a non-empty node, and the media type is specified, then decoding or dereferencing of the instance node prior to rendition is performed by datatype as follows:

- If the instance node either is of type or is derived from type `xsd:base64Binary`, then the data is base-64 decoded.
- If the instance node either is of type or is derived from type `xsd:hexBinary`, then the data is hex-binary decoded.

- If the instance node either is of type or is derived from type `xsd:anyURI`, then the data is treated as a URI and dereferenced.
- If the instance node is of any other type, then the data is used without modification.

If the `output` rendition is based on the `value` attribute, then the rendition is updated if the nodes referenced by the `value` expression change or if the content of any of the referenced nodes changes. Otherwise, the rendition of an `output` is updated if the node referenced by the Single-Node Binding changes, if the content of the referenced node changes, or if the media type changes. The media type can change by a change to the `mediatype` element's referenced node or its content (a host language may also allow DOM mutation of the content of the `mediatype` attribute or element). A change to the label associated with the `output` causes an update to the rendition of the label (which may affect the layout position of the main output content).

Failure to render the content indicated by the `output` element [should](#) result in an [xforms-output-error](#), a non-fatal error that does not halt XForms processing. Failures can occur on initial creation of the `output` or during user interface refresh (see Section [4.3.4 The xforms-refresh Event](#)). Failures can occur for many reasons, including

- Data to be decoded does not conform to the format of `xsd:base64Binary` or `xsd:hexBinary`
- An error dereferencing the URI in a node of or derived from type `xsd:anyURI`
- A data format error (e.g. invalid or unsupported image format)
- An unrecognized media type identifier string

The content model for the `output` element includes [UI Common](#) in order to allow action handlers for the [xforms-output-error](#) as well as to allow more comprehensive behavior and information to be provided for the `output`, e.g. via the `hint` element.

Examples:

<p>Explanatory Message</p> <pre>I charged you - <output ref="order/totalPrice"/> - and here is why:</pre>
<p>A graphical browser might render an output form control as follows:</p> <p>I charged you 100.0 - and here is why:</p> <ul style="list-style-type: none"> • Hidden Shipping charges • Expired discounts
<p>Displaying an image uploaded to instance data</p> <pre><xforms:model> <xforms:instance xmlns=""> <data></data> </xforms:instance> <xforms:bind nodeset="/data" type="xsd:base64Binary"/> </xforms:model></pre> <p>Given the <code>model</code> above, the following controls can upload an image to instance data and display it:</p> <pre><xforms:upload ref="/data" mediatype="image/*"> <xforms:label>Press me to attach a picture</xforms:label> </xforms:upload> <xforms:output ref="/data" mediatype="image/*"> <xforms:hint>This is the image you attached to the form.</xforms:hint> <xforms:message ev:event="xforms-output-error">Error attaching image data.</xforms:message> </xforms:output></pre>
<p>Output of node bound to <code>xsd:date</code></p> <pre><bind nodeset="birthdate" type="xsd:date" /> ... <output ref="birthdate"> <label>Lexical: </label> </output> <output ref="birthdate" appearance="full"> <label>Full: </label> </output> <output ref="birthdate" appearance="minimal"> <label>Minimal: </label> </output></pre>

A graphical browser may take into account the `appearance` and the localization information from the host language and present the above output form controls as follows:

Lexical: 1998-01-19 Full: 19 janvier 1998 Minimal: 19/01/1998

8.1.5.1 The *mediatype* Element (for output)

Binding attributes on author-optional element `mediatype` specify the location in the instance of the string that indicates the desired media type rendition for the parent `output`. If the binding attributes are not used, the `value` attribute must be used instead to specify the desired media type rendition.

Common Attributes: [Common](#), [Single Node Binding](#)

Special Attributes:

value

An XPath expression to be evaluated. The string result of the evaluation is used to specify the desired media type for output rendition. If a single node binding is expressed, then this attribute has no effect. The evaluation context is the same as would be applied to the evaluation of the single node binding. An empty string is used if the XPath evaluation fails.

8.1.6 The upload Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

mediatype

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

incremental

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

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

Note:

This control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

Implementation Requirements: For base64Binary or hexBinary data binding:

- When bound to an instance data node of type `xsd:base64Binary`, `xsd:hexBinary`, or a type derived by restriction thereof, on activation `upload` places the binary content in the content of the node with the indicated encoding.

Implementation Requirements: For anyURI data binding:

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

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

Note:

Implementors note that `upload` must associate the binary content, `mediatype`, and filename with that URI for [11.9.6 Serialization as multipart/related](#) and [11.9.7 Serialization as multipart/form-data](#) serialization.

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

Implementation Requirements: For all data bindings:

- Implementations with specific pen/digitizer hardware should (and implementations with other pointing devices may) support *scribble*—allowing in-place creation of pen-based data.
- Implementations with specific audio recording capabilities should support *record audio*—in-place recording of an audio clip.
- Implementations with a digital camera, scanner interface or screen capture should support *acquire image*—in-place upload of images from an attached device.
- Implementations with video recording capability should provide a *record video* option.
- Implementations with 3d capabilities should provide a 3d interface option.
- Implementations may provide proprietary implementations (for example, a `mediatype` of `text/rtf` could invoke an edit window with a proprietary word processing application)
- Implementations are encouraged to support other input devices not mentioned here.
- Implementations which cannot support upload for the given `mediatype` must make this apparent to the user.

See the child elements `filename` ([8.1.6.1 The filename Element](#)) and `mediatype` ([8.1.6.2 The mediatype Element \(for upload\)](#)).

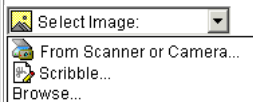
Example:

Uploading An Image

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


</upload>

A graphical browser might render this form control as follows:



Implementation Requirements:

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

8.1.6.1 The filename Element

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

Common Attributes: [Common](#), [Single Node Binding](#)

Content: EMPTY

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

Example:

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

8.1.6.2 The mediatype Element (for upload)

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

Common Attributes: [Common](#), [Single Node Binding](#)

Content: EMPTY

8.1.7 The range Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

start

Author-optional attribute containing a hint for the lexical starting bound for the range—a legal value for the underlying data. If provided, this value is used to further refine the constraints specified by the underlying model.

end

Author-optional attribute containing a hint for the ending bound for the range—a legal value for the underlying data. If provided, this value is used to further refine the constraints specified by the underlying model.

step

Author-optional attribute containing a delta-value to use for incrementing or decrementing the value. Must be of a type capable of expressing the difference between two legal values of the underlying data.

incremental

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

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

Note:

The above list of datatypes includes by derivation all of the integer datatypes (), all of the XForms datatypes defined in Section [5.2 XForms](#)

Datatypes that correspond to the allowed XML schema datatypes, and the datatypes defined in [5.2.4 xforms:dayTimeDuration](#) and [5.2.5 xforms:yearMonthDuration](#).

Note:

This control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

Implementation Requirements: Must allow input of a value corresponding to the bound datatype. Implementations should inform the user of the upper and lower bounds, as well as the step size, if any. If the instance data value is outside the upper or lower bounds, this form control must indicate an out-of-range condition. In graphical environments, this form control may be rendered as a "slider" or "rotary control".

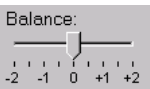
In the event of overlapping restrictions between the underlying datatype and the `start` and `end` hints, the most restrictive range should be used.

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

Examples:

Picking From A Range

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

A graphical browser might render this as follows:


Picking a date from a range

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

8.1.8 The trigger Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#) (author-optional)

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

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

Although a `trigger` element receives events associated with model item properties of a bound node, such as `xforms-readonly` and `xforms-invalid`, the XForms processor must not impart special behaviors on this control for model item properties other than the model item property `relevant` of a bound data node. For example, the `readonly` model item property of a bound data node does not affect whether or not the `trigger` can be activated.

Typically, a style sheet would be used to determine the exact appearance of form controls, though a means is provided to suggest an appearance through attribute `appearance`. Suggested renditions for the pre-defined values of this attribute are as follows:

"full": visually rendered as a standard button control with border.

"compact": visually rendered as a standard button control without border

"minimal": rendered with no border, a transparent background and underline font effect. This rendition hint is meant to be analogous to the typical visual rendition of an XHTML anchor element.

Example:

Simple Trigger

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

8.1.9 The submit Element

Description: This form control initiates a submission .

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#) (author-optional)

Special Attributes:

submission

Author-optional attribute containing a reference to element `submission`. If this attribute is given but does not identify a `submission` element,

then activating the `submit` does not result in the dispatch of an `xforms-submit` event. If this attribute is omitted, then the first `submission` in document order from the `model` associated with the in-scope evaluation context is used.

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

Implementation Requirements: The default action for event `DOMActivate` is to dispatch event `xforms-submit` to the `submission` element specified by attribute `submission` (or its default). Upon activation, this control must become unavailable for further activations until the submit process concludes with either an `xforms-submit-done` or `xforms-submit-error` event.

Typically, a style sheet would be used to determine the exact appearance of form controls, though a means is provided to suggest an appearance through attribute `appearance`. Suggested renditions for the pre-defined values of this attribute are the same as for [trigger](#).

Example:

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

8.1.10 The select Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

selection

Author-optional attribute determining whether free entry is allowed in the list. Default is "closed".

incremental

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

Data Binding Restrictions: any simpleContent capable of holding a sequence. The restriction to binding simpleContent exists when the choices are authored as part of the user interface control as shown in this section. Element `itemset` (described in [9.3.6 The itemset Element](#)) creates dynamic selection items and allows the available choices to be obtained from an XForms Model. When `itemset` uses the `value` element, the restriction to binding simpleContent remains in effect. However, the `itemset` also allows for the selection and deselection of subtrees of instance data using the `copy` element, and when using that construct, the data binding restriction to simpleContent is relaxed, but the form control must bind to an element with no mixed content.

Note:

Except in the case described above where the simpleContent data binding restriction is relaxed, this control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

Note:

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

```
Incorrect Type Declaration
<item>
  <value>United States of America</value>
  ...
</item>
```

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

Implementation Requirements: The label for each choice must be presented, and the control must allow any number of selections, possibly none. When this form control uses the `value` element for selection, it stores the values corresponding to the selected choices in a space separated list in the location addressed by the binding attributes. The values to be stored for selected items are either directly specified as the contents of element `value`, or specified indirectly through binding attributes on element `value`. When this form control uses the `copy` element for selection, it stores copies of the subtrees corresponding to the selected choices in the location addressed by the binding attributes.

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

For closed selections: If the instance data matches the storage data of one or more of the selection items, those items are selected. If there is no match, no items are initially selected. If any of the stored values or subtree copies do not correspond to an item with a matching storage value or subtree, the form control must indicate an out-of-range condition. If the form control switches to or from being out-of-range, then `xforms-out-of-range` or `xforms-in-range` must be dispatched to the form control.

For open selections: When using dynamic selections with the `itemset` and `copy` elements, open selection has no effect. If the instance data matches the storage values specified by one or more of the selection items, then all such matching items are selected. If any instance data list values do not match the storage value specified by one or more of the items, all such non-matching values are retained, as if entered through

free entry. Free entry text is handled the same as form control `input` ([8.1.2 The input Element](#)), possibly in multiplicity.

For both closed and open selections, any selection item with an empty storage data subtree or a storage value that is either empty or contains only white space characters must remain deselected.

For both closed and open selections, the above rules describe which items are considered to be selected and deselected by the control. The `select` form control changes the states of selected and deselected items on creation, refresh, and user selection or deselection of an item. Newly selected items receive the event `xforms-select` immediately after all newly deselected items receive the event `xforms-deselect`. The content of the instance node bound to the selection control must only be changed by the addition or deletion of storage data associated with items that have been selected or deselected. Content not associated with selection items is preserved. For selection controls that use the `value` element, the net effect of newly selected and deselected items is computed into a string, preserving content not associated with selection items, and the result is then committed to the bound instance node by using the XForms Action [10.2 The setvalue Element](#). For selection controls that use the `copy` element, the individual subtrees associated with the newly selected and deselected items are added or removed individually by using [10.3 The insert Element](#) and [10.4 The delete Element](#).

Implementation Hints:

For closed selections, when the form control is created or refreshed to reflect bound instance data, behavior equivalent to the following steps occurs:

1. The content parts (space-separated values or subtree copies) in the bound instance data node are compared with the form control items' storage data (values or subtree copies).
2. Each item with storage data (value or subtree copy) equal to an instance data content part becomes selected if it was not already selected.
3. Each item with storage data missing from the instance data content becomes deselected if it was not already deselected.
4. If there are instance data content parts for which there is no corresponding selection item, the form control indicates an out-of-range condition.

When the user selects an item which was previously deselected, behavior equivalent to the following steps occurs:

1. If the item's storage data (value or subtree copy) was not present in the bound instance data, the item's storage data is inserted into the instance data content list. The exact location of the insertion is implementation-dependent. Any other item having the same storage data becomes selected as well.
2. If the item's storage data was already present in the bound instance data, the bound instance data is left unchanged.

When the user deselects an item which was previously selected, behavior equivalent to the following steps occurs:

1. If the item's storage data was present in the bound instance data, the item's storage data is removed from the instance data content list. Any other item having the same storage data becomes deselected as well.
2. If the item's storage data was already absent from the bound instance data, the bound instance data is left unchanged.

For open selections: when the form control is created or refreshed to reflect bound instance data, the behavior is the same as with closed selection, except the form control never indicates an out-of-range condition.

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.

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

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

Example:

Selecting Ice Cream Flavor

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

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

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

<code>appearance="full"</code>	<code>appearance="compact"</code>	<code>appearance="minimal"</code>

8.1.11 The select1 Element

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

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#)

Special Attributes:

selection

Author-optional attribute determining whether free entry is allowed in the list. Default is "closed".

incremental

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

Data Binding Restrictions: Binds to any `simpleContent`. The restriction to binding `simpleContent` exists when the choices are authored as part of the user interface control as shown in this section. Element `itemset` (described in [9.3.6 The itemset Element](#)) creates dynamic selection items and allows the available choices to be obtained from an XForms Model. When `itemset` uses the `value` element, the restriction to binding `simpleContent` remains in effect. However, the `itemset` also allows for the selection and deselection of subtrees of instance data using the `copy` element, and when using that construct, the data binding restriction to `simpleContent` is relaxed, but the form control must bind to an element with no mixed content.

Note:

Except in the case described above where the `simpleContent` data binding restriction is relaxed, this control cannot bind to element nodes that have element children. See [8.1.1 Implementation Requirements Common to All Form Controls](#) for user interface processing rules common to all form controls.

Implementation Requirements: The label for each choice must be presented, and the control must allow at all times exactly one selection. When this form control uses the `value` element for selection, it stores the value corresponding to the selected choice in the location addressed by the binding attributes. The value to be stored is either directly specified as the contents of element `value`, or specified indirectly through binding attributes on element `value`. When this form control uses the `copy` element for selection, it stores a copy of the subtree corresponding to the selected choice in the location addressed by the binding attributes.

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

For closed selections: If the instance data matches the storage data of one of the selection items, that item is selected. If there is no match, no items are initially selected. If there is no match and the storage data is non-empty, the form control must indicate an out-of-range condition. If the form control switches to or from being out-of-range, then `xforms-out-of-range` or `xforms-in-range` must be dispatched to the form control.

For open selections: When using dynamic selections with the `itemset` and `copy` elements, open selection has no effect. If the instance data matches the storage value specified by one of the selection items, then the first such matching item is selected. Otherwise, no items are selected and the instance data value is retained, as if entered through free text entry. Free entry text is handled the same as form control `input` ([8.1.2 The input Element](#)).

For both closed and open selections, any selection item with a storage value which is empty or which contains only white space characters must remain deselected.

For both closed and open selections, the above rules describe which items are considered to be selected by the control. Items that are not selected are considered to be deselected. The `select1` form control changes the states of selected and deselected items on creation, refresh, and user selection or deselection of an item. A newly selected item receives the event `xforms-select` immediately after all other items receive the event `xforms-deselect`. The content of the instance node bound to the selection control must only be changed by the addition or deletion of storage data associated with items that have been selected or deselected. Content not associated with selection items is preserved. For selection controls that use the `value` element, the net effect of newly selected and deselected items is computed into a string, preserving content not associated with selection items, and the result is then committed to the bound instance node by using the XForms Action [10.2 The setValue Element](#). For selection controls that use the `copy` element, the individual subtrees associated with the newly selected and deselected items are added or removed individually by using [10.3 The insert Element](#) and [10.4 The delete Element](#).

Implementation Hints:

For closed selections, when the form control is created or refreshed to reflect bound instance data, behavior equivalent to the following steps occurs:

1. The bound instance data is compared with the items' storage data (values or subtree copies).
2. If no item with storage data (value or subtree copy) in the bound instance data node is selected, then the first item with storage data in the instance data content, if any, becomes selected. Otherwise, if an item with storage data in the bound instance data node is selected, then the first selected item remains selected, and any other items with storage data matching the selected item are deselected.
3. If there is a selected item, then all items with storage data not equal to the selected item are deselected, and their representative storage data is removed from the bound instance node content.
4. If no item has storage data in the bound instance data node content and the instance data node content is not empty, then the form control

indicates an out-of-range condition.

When the user selects an item which was previously deselected, behavior equivalent to the following steps occurs:

1. All selected items other than the newly selected item are deselected, if any, and the storage data of any deselected items whose storage data does not match the newly selected item are removed from the bound instance node data.
2. The newly selected item becomes selected. If its storage data (value or subtree copy) is not present in the bound instance data, then the item's storage data is inserted into the instance data. The exact location of the insertion is implementation-dependent, but the newly inserted data is not accompanied by any other data unless the data does not match any items for the selection control.

When the user deselects an item which was previously selected, behavior equivalent to the following steps occurs:

1. The item is deselected and its storage data is removed from the bound instance node data.
2. If the bound instance data node is not empty, then the form control indicates an out-of-range condition.

For open selections: when the form control is created or refreshed to reflect bound instance data, the behavior is the same as with closed selection, except the form control never indicates an out-of-range condition.

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.

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

Example:

Pick A Flavor

```

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

```

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

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

appearance="full"

Flavour
☐ Vanilla
 ☒ Strawberry
 ☐ Chocolate

appearance="compact"

Flavour

Vanilla
Strawberry
Chocolate

appearance="minimal"

Flavour

Vanilla

8.2 Common Support Elements

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

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

8.2.1 The label Element

This element provides a descriptive label for the containing form control. The descriptive label can be presented visually and made available to accessibility software so the visually-impaired user can obtain a short description of form controls while navigating among them.

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

Special Attributes: None

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

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

8.2.2 The help Element

The author-optional element `help` provides a convenient way to attach help information to a form control.

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

Special Attributes: None

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

An example of this element is at [10.16 The message Element](#).

8.2.3 The hint Element

The author-optional element `hint` provides a convenient way to attach hint information to a form control.

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

Special Attributes: None

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

An example of this element is at [10.16 The message Element](#).

8.2.4 The alert Element

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

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

Special Attributes: None

The message specified can exist in instance data or as inline text. If more than one source of message is specified in this element, the order of precedence is: single node binding attributes, inline text. See [G XForms and Styling](#) for examples to see how this might be presented to the user.

8.3 Common Markup for Selection Controls

8.3.1 The choices Element

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

Common Attributes: [Common](#)

8.3.2 The item Element

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

Common Attributes: [Common](#)

8.3.3 The value Element

This element provides a storage value to be used when an `item` is selected. The storage value is determined by one of three methods, in order of precedence:

1. the value of a node indicated by a single node binding expression, if specified
2. the result of evaluating an XPath expression appearing in attribute `value`, if specified
3. the inline content of the `value` element (when neither the single node binding nor the `value` attribute are expressed).

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

Special Attributes:

value

Author-optional. An XPath expression to be evaluated. The string result of the evaluation is used as the storage value of the `item` when it is selected. If a single node binding is expressed, then this attribute has no effect. The evaluation context is the same as would be applied to the evaluation of the single node binding. An empty string is used if the XPath evaluation fails.

Data Binding Restriction: All lexical values must be valid according to the datatype bound to the selection control. If the single node binding attributes are used and indicate a node in a model other than the bound node of the containing selection control, then an `xforms-binding-exception` must occur.

9 Container Form Controls

This chapter covers XForms view layer features for combining [core form controls](#) into user interfaces using [container form controls](#). All core form controls defined in [8 Core Form Controls](#) are treated as individual units for purposes of visual layout e.g., in XHTML processing. A **container form control** is a form control that provides the ability to combine other form controls in its content into user interfaces.

Aggregation of form controls with markup defined in this chapter provides semantics about the relationship among user interface controls; such

knowledge can be useful in delivering a coherent UI to small devices. For example, if the user interface needs to be split up over several screens, controls appearing inside the same aggregation would typically be rendered on the same screen or page.

9.1 The XForms Group Module

A `group` element is a [container form control](#) that allows a form author to aggregate other form controls into a single, aggregate user interface component. The elements and attributes included in this module are:

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

9.1.1 The group Element

The `group` element is used as a container for defining a hierarchy of form controls. Groups can be nested to create complex hierarchies. The author-optional `label` element has special significance when it appears as the first element child of `group`, representing a label for the entire group.

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#) (author-optional)

Although a `group` element receives events associated with model item properties of a bound node, such as `xforms-readonly` and `xforms-invalid`, no special behavior is imparted by the `group` onto the content elements in the group as a direct result of any model item property. The model item property `relevant` of a bound data node can indirectly affect the content of the group via its contribution to deciding whether the `group` is relevant or non-relevant. A group is considered to be non-relevant if and only if:

- the `Single Node Binding` is expressed and resolves to empty nodeset,
- the `Single Node Binding` is expressed and resolves to a non-relevant instance node,
- the `group` is contained by a non-relevant `switch` or `group` (which includes a non-relevant `repeat` object), or
- the `group` is contained by a non-selected `case` element of a `switch`.

All content elements (e.g. core form controls, groups, switches, repeats and host language content) within a non-relevant group are handled as non-relevant. When a `group` becomes non-relevant, it must receive event `xforms-disabled` and then the XForms action handlers that are listening for events on the non-relevant `group` must be disabled. When a non-relevant `group` changes to being relevant, the XForms action handlers that listen for events on the `group` must become enabled and then the `group` must receive the event `xforms-enabled`.

Note:

If a group is non-relevant, then the rendering approach used to signify non-relevance is applied to the entire content of the group.

Example:

Grouping Related Controls	
<pre><group ref="address"> <label>Shipping Address</label> <input ref="line_1"> <label>Address line 1</label> </input> <input ref="line_2"> <label>Address line 2</label> </input> <input ref="postcode"> <label>Postcode</label> </input> </group></pre>	

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

9.2 The XForms Switch Module

A `switch` element is a [container form control](#) that allows the creation of user interfaces where the user interface can be varied based on user actions and events. The elements and attributes included in this module are:

Element	Attributes	Minimal Content Model
switch	Common , UI Common , Single Node Binding (author-optional)	case+
case	Common , <code>selected (xsd:boolean)</code>	label?, ((Core Form Controls) group switch repeat Action)*
toggle	Common , Events , Action Common , <code>case (xsd:IDREF)</code>	case?

9.2.1 The switch Element

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

Note:

This is separate from XForms `relevant` processing (see [6.1.4 The relevant Property](#)), which is based on the current state of the XForms Model. As an example, portions of a questionnaire pertaining to the user's automobile may become relevant only if the user has answered in the affirmative to the question 'Do you own a car?'.

Common Attributes: [Common](#), [UI Common](#), [Single Node Binding](#) (author-optional)

Although a `switch` element receives events associated with model item properties of a bound node, such as `xforms-readonly` and `xforms-`

invalid, no special behavior is imparted by the `switch` onto the content elements in the selected `case` as a direct result of any model item property. The model item property `relevant` of a bound data node can indirectly affect the content of the selected `case` via its contribution to deciding whether the `switch` is relevant or non-relevant. The non-relevance of a `switch` is determined in the same way as it is for `group` and similarly applies to the entire content. Also, as with `group`, when a `switch` becomes non-relevant, it must receive event `xforms-disabled` and then the XForms action handlers that are listening for events on the non-relevant `switch` must be disabled. As well, when a non-relevant `switch` changes to being relevant, the XForms action handlers that listen for events on the `switch` must become enabled and then the `switch` must receive the event `xforms-enabled`.

Example:

```
switch
<switch>
  <case id="in" selected="true">
    <input ref="yourname">
      <label>Please tell me your name</label>
      <toggle ev:event="DOMActivate" case="out"/>
    </input>
  </case>
  <case id="out" selected="false">
    <html:p>Hello <output ref="yourname" />
    <trigger id="editButton">
      <label>Edit</label>
      <toggle ev:event="DOMActivate" case="in"/>
    </trigger>
  </html:p>
  </case>
</switch>
```

The above results in the portion of the user interface contained in the first `case` being displayed initially. This prompts for the user's name; filling in a value and *activating* the control e.g., by pressing `enter` results switches to the alternate case, with a read-only `output` rendering. Activating the trigger labeled "Edit" in turn switches back to the original case.

9.2.2 The case Element

This element encloses markup to be conditionally rendered. The content elements (e.g. form controls, groups, switches, repeats and host language elements) within a non-selected `case` behave as if they were in a non-relevant `group` (see [9.1.1 The group Element](#)). Similarly, content elements in a `case` that becomes selected behave as if they were in a `group` that has become relevant. The attribute `selected` determines the initial selected state.

Common Attributes: [Common](#)

Special Attributes:

selected

Author-optional selection status for the case. The default value is "false".

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

9.3 The XForms Repeat Module

The XForms specification allows the definition of repeating structures such as multiple items within a purchase order. When defining the XForms Model, such higher-level collections are constructed out of basic building blocks; similarly, this section defines a [container form control](#) called `repeat` that can bind to data structures such as lists and collections and provide a user interface for each node of the list or collection.. The elements and attributes included in this module are:

Element	Attributes	Minimal Content Model
repeat	Common , UI Common , Node Set Binding , <code>startindex</code> (xsd:positiveInteger), <code>number</code> (xsd:nonNegativeInteger)	((Core Form Controls) <code>group</code> <code>switch</code> <code>repeat</code> Action)*
setindex	Common , Events , Action Common , <code>repeat</code> (xsd:IDREF), <code>index</code> (number XPath Expression)	EMPTY
itemset	Common , Node Set Binding	<code>label</code> , (<code>value</code> <code>copy</code>), (UI Common)*
copy	Common , Single Node Binding (author-optional)	EMPTY
(various)	[<code>repeat-nodeset</code> , <code>repeat-bind</code> , <code>repeat-model</code>] (Node Set Binding attributes), <code>repeat-startindex</code> (xsd:positiveInteger), <code>repeat-number</code> (xsd:nonNegativeInteger)	N/A

9.3.1 The repeat Element

This element defines a UI mapping over a node-set selected by Node Set Binding Attributes. This node-set is called a **repeat collection**.

For example:

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

Common Attributes: [Common](#), [UI Common](#), [Node Set Binding](#)

Special Attributes:

startindex

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

number

Author-optional hint to the XForms Processor as to how many elements from the collection to display.

This element operates over a [repeat collection](#) by binding the encapsulated user interface controls to each element of the collection. If an element of the collection is non-relevant, then the rendering approach used to signify non-relevance is applied to the associated user interface controls. Attributes on this element specify how many members of the collection are presented to the user at any given time. XForms Actions `insert`, `delete`, and `setindex` can be used to operate on the collection—see [10 XForms Actions](#). Another way to view repeat processing (disregarding special user interface interactions) is to consider "unrolling" the repeat. The above example is similar to the following (given four `item` elements in the returned node-set):

Repeat Unrolled

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

Repeat Collection

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

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

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

9.3.2 Nested Repeats

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

It is possible to nest repeat elements to create more powerful user interface for editing structured data. [H.2 Editing Hierarchical Bookmarks Using XForms](#) is an example of a form using nested repeats to edit hierarchical data consisting of bookmarks within multiple sections. Consider the following `insert` statement that appears as part of that example.

Repeat Index and Nested Repeats

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

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

9.3.3 Repeat Processing

The markup contained within the body of element `repeat` specifies the user interface to be generated for each [repeat item](#) of the underlying repeat collection. During user interface initialization (see [4.2.2 The xforms-model-construct-done Event](#)), the following steps are performed for `repeat`:

1. The Node Set Binding is evaluated to locate the [repeat collection](#) to be operated on by this `repeat`.
2. The `index` for this repeating structure is initialized to the value of `startindex`. If the initial `startindex` is less than 1 it defaults to 1. If the index is greater than the initial node-set then it defaults to the size of the node-set.
3. User interface as specified by the `repeat` is generated for the requisite number of members of the collection as specified by attributes on element `repeat`.

For each node of the repeat collection, a **repeat item** is defined to be the aggregation of the node, its position, the size of the repeat collection, and a [repeat object](#).

A **repeat object** is an implicitly generated `group` element that contains the set of run-time objects generated to represent the repeat template content for a single repeat item of the [repeat collection](#). These run-time objects are form controls, XForms actions and other host language elements that correspond to elements in the repeat template.

Note:

The capture and bubble phases of events dispatched to the run-time objects behave as if the repeat object were a child of element `repeat`. The repeat template content, including action handlers, is made unavailable to the host language processor. Copies of the repeat template content, including Action handlers, are made available via the repeat objects.

Example: Handling an Event Dispatched to a `repeat` Element

```
<repeat id="X" ... >
  ...
</repeat>

<action ev:event="xforms-scroll-first" ev:target="X" ev:observer="X">
  ...
</action>
```

A new repeat item is created dynamically at any time in the lifecycle of the form (i.e. any time after `xforms-model-construct-done`) whenever a new node is added to the [repeat collection](#). There are many ways to add new nodes to a [repeat collection](#), including but not limited to the following:

- An `insert` action can add one or more nodes that match the repeat nodeset;
- The new instance data subtree created by a `submission` instance replacement may contain nodes that match the repeat nodeset;
- A `setvalue` action or a `calculate` may change a value that causes one or more nodes to match the repeat nodeset.

Any time a new repeat item is created, XML Event handlers declared within the corresponding repeat object are initialized, and the user interface form controls generated for the repeat object are initialized in the same manner as the user interface initialization that is performed during default processing of `xforms-model-construct-done`. For example, if the repeat object contains an inner `repeat` run-time object, then it is initialized according to the list of steps at the beginning of this section ([9.3.3 Repeat Processing](#)).

The processing model for repeating structures includes an `index` that points to the *current* repeat item in the [repeat collection](#). This repeat index is accessed via XForms function `index` ([7.7.5 The index\(\) Function](#)) and manipulated via XForms Action `setindex` ([10.5 The setindex Element](#)). This index can be used as a reference point for `insert` and `delete` actions. 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.

If one or more nodes have been added to the repeat collection by an `insert` action, then the repeat items corresponding to the new nodes must be created and initialized, and the repeat index must be updated to indicate the repeat item corresponding to the last node added by the `insert`.

Note:

The change of index on a `repeat` does not cause the index of any `repeat` nested within it to be re-initialized.

The repeat item generation and repeat index update on insertion must behave as if it occurs in response to the `xforms-insert` event dispatched by the `insert` action. The index update must behave as if it occurs when the `xforms-insert` event reaches the parent of the target `instance` element in the capture phase.

Note:

An event handler that listens for `xforms-insert` on `instance` in the default phase has access to the updated index value via function `index()`.

A repeat item can also be destroyed dynamically at any time in the lifecycle of the form whenever a node is removed from the [repeat collection](#). When a repeat item is destroyed, the repeat object and all of its inner form controls are eliminated, including inner repeats, switches and groups, and all XML Event handlers created by the repeat object are eliminated. There are many ways to remove repeat items from a [repeat collection](#), including but not limited to the following:

- A `delete` action can remove nodes that matched the repeat nodeset;
- The new instance data subtree created by a `submission` instance replacement may replace nodes that matched the repeat nodeset;
- A `setvalue` action or a `calculate` may change a value that causes one or more nodes to stop matching the repeat nodeset.

If one or more nodes have been removed from the repeat collection by a `delete` action, then the repeat items corresponding to the deleted nodes must be destroyed and the repeat index must be updated based on the rules below.

1. If, prior to node deletion, the repeat index indicated a repeat that is still contained in the repeat collection after node deletion, then the index is adjusted, if necessary, to indicate that same repeat item.
2. Otherwise, if all repeat items in the collection have been destroyed, the repeat index is set to 0.
3. Otherwise, if the repeat index was pointing to one of the deleted repeat items, and if the new size of the collection is smaller than the index, the index is changed to the new size of the collection.
4. Otherwise, if the repeat index was pointing to one of the deleted repeat items, and if the new size of the collection is equal to or greater than the index, the index is not changed.

Note:

The change of index on a `repeat` does not cause the index of any `repeat` nested within it to be re-initialized.

The repeat index update on deletion behaves as if it occurs in response to the `xforms-delete` event dispatched by the `delete` action. Specifically, the index update behaves as if it occurs when the `xforms-delete` event reaches the parent of the target `instance` element in the capture phase.

9.3.4 User Interface Interaction

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

Notice that the markup encapsulated by element `repeat` acts as the template for the user interface that is presented to the user. As a consequence, statically authored `IDREF` attributes must be interpreted based on a combination of repeat indexes and where the `IDREF` attributes appear relative to the element bearing the matching ID. Based on the `IDREF` resolution rules given in [4.7 Resolving ID References in XForms](#), it is possible to toggle the `case` of a `switch` even when it is within one or more `repeat` elements. Similarly, it is possible to set the focus to controls and dispatch events to elements that are within one or more `repeat` elements.

If the focus is transferred to a form control within a `repeat` by any means, such as by an XForms action or by user interaction, the index of the `repeat` is updated to indicate the item of the [repeat collection](#) that contains the control. If the repeat item containing the focused control contains any inner repeat objects, their indexes are not changed. However, the repeat index update is recursive for all outer repeats that contain the focused control; the index of each outer containing `repeat` is adjusted appropriately. These changes of repeat index occurs as if by invoking the `setindex` action.

9.3.5 Creating Repeating Structures Via Attributes

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

Tables And Repeating Structures

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

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

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

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

Tables And Repeating Structures

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

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

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

9.3.6 The itemset Element

This element allows the creation of dynamic selections within controls `select` and `select1`, where the available choices are determined at run-time. The node-set that holds the available choices is specified via the Node Set Binding. Child elements `label` and `value` indirectly specify the label and storage values. Notice that the run-time effect of `itemset` is the same as using element `choices` with child `item` elements to statically author the available choices.

For each node of the Node Set Binding, an associated `item` element is created. XForms Actions appearing in the content of an `itemset` are created within each `item` element, and the in-scope evaluation context for these XForms Actions is based on the node for which the `item` was generated as described in Section [7.2 Evaluation Context](#).

Note:

As with the `repeat` element, the `itemset` template content, including XForms Actions, is made unavailable. Copies of the `itemset` template content, including XForms Actions, are made available via repeated `item` objects.

Common Attributes: [Common](#), [Node Set Binding](#)

Note:

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

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

```
Dynamic Choice Of Ice Cream Flavors

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

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

9.3.7 The copy Element

Structurally, this element is similar to [8.3.3 The value Element](#). It differs in that it can only be used within `itemset`, and that it works with subtrees of instance data rather than simple values.

Common Attributes: [Common](#), [Single Node Binding](#) (author-optional)

If the single node binding attributes indicate a node in a model other than the bound node of the containing selection control, then an `xforms-binding-exception` must occur.. When a `copy` item is selected, the following rules apply:

- The target node, selected by the binding attributes on the list form control, must be an element node, otherwise an exception results ([4.5.1 The xforms-binding-exception Event](#)).
- The element node associated with the `item`, selected by the binding attributes on `copy`, is deep copied as a child of the target node by using an `insert` action ([10.3 The insert Element](#)).
- A full computational dependency rebuild is done, followed by recalculate, revalidate, and refresh.

When a `copy` item is deselected, the following rules apply:

- The target node, selected by the binding attributes on the list form control, must be an element node, otherwise an exception results ([4.5.1 The xforms-binding-exception Event](#)).

- The child element node associated with the `item`, selected by the binding attributes on `copy`, is deleted by using a `delete` action ([10.4 The delete Element](#)).
- A full computational dependency rebuild, followed by recalculate, revalidate, and refresh.

Note:

If the target node of the `select` or `select1` is readonly, then the insertion or deletion associated with the copy operation is not performed.

10 XForms Actions

This chapter defines the controller layer of XForms, an XML Events-based [XML Events](#) common set of actions that can be invoked in response to events.

Note:

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

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

The elements and attributes included in this module are:

Element	Attributes	Minimal Content Model
action	Common , Events , Action Common	(Action)*
setvalue	Common , Events , Action Common , Single Node Binding , value (string XPath Expression)	PCDATA
insert	Common , Events , Action Common , Node Set Binding , context (node XPath Expression), at (number XPath Expression), position ("before" "after"), origin (nodeset XPath Expression)	EMPTY
delete	Common , Events , Action Common , Node Set Binding , context (node XPath Expression), at (number XPath Expression)	EMPTY
setindex	Common , Events , Action Common , repeat (xsd:IDREF), index (number XPath Expression)	EMPTY
toggle	Common , Events , Action Common , case (xsd:IDREF)	case?
setfocus	Common , Events , Action Common , control (xsd:IDREF)	control?
dispatch	Common , Events , Action Common , name (xsd:NMTOKEN), targetid (xsd:IDREF), delay (xsd:nonNegativeInteger), bubbles (xsd:boolean), cancelable (xsd:boolean)	name?, targetid?, delay? [in any order]
rebuild	Common , Events , Action Common , model (xsd:IDREF)	EMPTY
recalculate	Common , Events , Action Common , model (xsd:IDREF)	EMPTY
revalidate	Common , Events , Action Common , model (xsd:IDREF)	EMPTY
refresh	Common , Events , Action Common , model (xsd:IDREF)	EMPTY
reset	Common , Events , Action Common , model (xsd:IDREF)	EMPTY
load	Common , Events , Action Common , Single Node Binding (author-optional), resource (xsd:anyURI), show ("new" "replace")	resource?
send	Common , Events , Action Common , submission (xsd:IDREF)	EMPTY
message	Common , Events , Action Common , Single Node Binding (author-optional), level ("ephemeral" "modeless" "modal"), QNameButNotNCName	(PCDATA UI Inline)*

This module also defines the content set **"Action"**, which includes the following elements:

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

The following group of attributes, here called **Action Common**, are available to all [Action](#) elements:

Element	Attributes
Action	if (boolean XPath Expression), while (boolean XPath Expression)

if

Author-optional attribute defined in Section [10.17 Conditional Execution of XForms Actions](#).

while

Author-optional attribute defined in Section [10.18 Iteration of XForms Actions](#).

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

The following example shows how events can be used:

Action Syntax
<pre><xforms:trigger> <xforms:label>Reset</xforms:label></pre>


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

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

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

- Name
- Common Attributes
- Special Attributes
- Description of behavior

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

An **outermost action handler** is an action that is activated when the XForms processor is not executing any other action handlers.

An **inner action handler** is an action that is activated when the XForms processor is executing the declared actions of an [outermost action handler](#). An inner action handler may be within the content of the outermost action handler, or it may be executed as the response to an event dispatched while performing all of the actions initiated by the [outermost action handler](#).

Deferred Updates: Sequences of one or more XForms Actions have a deferred effect on XForms model and user interface processing. Implementations are free to use any strategy to accomplish deferred updates, but the end result must be as follows: Instance data changes performed by a set of actions do not result in immediate computation dependency rebuilding, recalculation, revalidate and form control refreshing until the termination of the [outermost action handler](#), as described here. Each XForms model can be thought of as having a set of deferred update Boolean flags, initially *false* at the start of an [outermost action handler](#), to indicate whether each of the actions *rebuild*, *recalculate*, *revalidate*, and *refresh* are required for that model upon termination of the [outermost action handler](#).

By default, the behavior of an action handler is performed one time when the action is encountered in the execution sequence. However, execution of an action handler may be conditional or iterated, as described in [10.17 Conditional Execution of XForms Actions](#) and [10.18 Iteration of XForms Actions](#).

Execution of an [outermost action handler](#) begins by setting the XForms processor into the state of executing an [outermost action handler](#). The [outermost action handler](#) is then performed, which may include the execution of [inner action handlers](#). Finally, the XForms processor is set into the state of not executing an [outermost action handler](#) and then the deferred update is performed for each model.

The **deferred update behavior** for a model consists of examining each deferred update Boolean flag in the order of *rebuild*, *recalculate*, *revalidate*, and *refresh*, and for each *true* flag, set the flag to *false* and then dispatch the proper event to the model for that deferred update flag (i.e. dispatch *xforms-rebuild* for a true *rebuild* flag, *xforms-recalculate* for a true *recalculate* flag, *xforms-revalidate* for a true *revalidate* flag, and *xforms-refresh* for a true *refresh* flag).

Note:

The XForms processor is not considered to be executing an [outermost action handler](#) at the time that it performs deferred update behavior for XForms models. Therefore, event handlers for events dispatched to the user interface during the deferred refresh behavior are considered to be new [outermost action handler](#).

Actions that manipulate properties of the XForms view layer begin by invoking the [deferred update behavior](#) so that the model and all data are up to date prior to performing the action. The XForms Actions in this category are:

```
setfocus
toggle
setindex
```

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

```
rebuild
recalculate
revalidate
refresh
```

Similarly, if the default processing of any of the events *xforms-rebuild*, *xforms-recalculate*, *xforms-revalidate*, or *xforms-refresh* are performed, then the corresponding deferred update flag is cleared. The XForms Actions that can dispatch these events are:

```
reset
dispatch
```

XForms Actions that change the tree structure of instance data result in setting all four deferred update flags to *true* for the model over which they operate. The XForms Actions in this category are:

```
insert
delete
```

XForms Actions that change only the value of an instance node results in setting the deferred update flags for *recalculate*, *revalidate*, and *refresh* to *true* and making no change to the deferred update flag for *rebuild* for the model over which they operate. The XForms Actions in this category are:

```
setvalue
setindex
```

Finally, the XForms *submission* process can affect deferred update behavior. See Section [11.2 The xforms-submit Event](#) for details. XForms actions that are capable of initiating an XForms submission are:

```
send
dispatch
```

10.1 The action Element

This action causes its child actions to be invoked in the order that they are specified in the document.

Common Attributes: [Common](#), [Events](#), [Action Common](#)

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

10.2 The setvalue Element

This action explicitly sets the value of the specified instance data node. This action has no effect if the Single Node Binding does not select an instance data node or if a readonly instance data node is selected. An `xforms-binding-exception` occurs if the Single Node Binding indicates a node whose content is not simpleContent (i.e., a node that has element children).

Common Attributes: [Common](#), [Events](#), [Action Common](#), [Single Node Binding](#)

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate, with the result stored in the selected instance data node. The evaluation context for this XPath expression is the result from the Single Node Binding. To obtain the value, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

The element content of `setvalue` specifies the literal value to set; this is an alternative to specifying a computed value via attribute `value`. If neither a `value` attribute nor text content are present, the effect is to set the value of the selected node to the empty string (""). If both are present, the `value` attribute is used. The following examples contrast these approaches:

setvalue with Expression
<pre><setvalue bind="put-here" value="a/b/c"/></pre>
This causes the string value at <code>a/b/c</code> in the instance data to be placed on the single node selected by the bind element with <code>id="put-here"</code> .
setvalue with Literal
<pre><setvalue bind="put-here">literal string</setvalue></pre>
This causes the value "literal string" to be placed on the single node selected by the bind element with <code>id="put-here"</code> .

Note:

See Section [7.10.4 The context\(\) Function](#) for an example in which the `context()` function is used to provide the same initial evaluation context node to both the `ref` and `value` attributes. See Appendix [B Patterns for Data Mutations](#) for numerous further usage patterns for `sevalue`, `insert` and `delete`.

All strings are inserted into the instance data as follows:

- Element nodes: If element child nodes are present, then an `xforms-binding-exception` occurs. Otherwise, regardless of how many child nodes the element has, the result is that the string becomes the new content of the element. In accord with the data model of [XPath 1.0](#), the element will have either a single non-empty text node child, or no children string was empty.
- Attribute nodes: The string-value of the attribute is replaced with a string corresponding to the new value.
- Text nodes: The text node is replaced with a new one corresponding to the new value, or the text node is eliminated if the new value is the empty string.
- Namespace, processing instruction, and comment nodes: behavior is undefined (implementation-dependent).
- the XPath root node: an `xforms-binding-exception` occurs.

Note:

This action affects [deferred updates](#) by setting the deferred update flags for recalculate, revalidate and refresh.

10.3 The insert Element

The `insert` action is used to create one or more nodes of instance data by cloning one or more existing instance nodes. Attributes of the `insert` action specify the node or nodes to be cloned and the location within instance data where the clones will appear. The clones are deep copies of the original nodes except the contents of nodes of type `xsd:ID` are modified to remain as unique values in the instance data after the clones are inserted.

Common Attributes: [Common](#), [Events](#), [Action Common](#), [Node Set Binding](#) (author-optional)

Special Attributes:

context

Author-optional attribute containing an XPath expression evaluated using the in-scope evaluation context. If the `model` attribute is present, then it is processed as described in [7.2 Evaluation Context](#) before evaluating this attribute. The Node Set Binding is required unless this attribute is present. The result of the XPath expression is used to override the in-scope evaluation context. If the result is an empty nodeset or not a nodeset, then the insert action is terminated with no effect. Otherwise, the first node of the nodeset is used as the new in-scope evaluation context node, and the context position and size are set to 1. By adjusting the in-scope evaluation context, this attribute affects the subsequent evaluation of many other attributes that can appear on `insert`, including `if`, `while`, `nodeset` and `origin`.

origin

Author-optional attribute containing an XPath expression evaluated using the in-scope evaluation context, which may have been amended by the `context` attribute. The `origin node-set` is the set of one or more nodes to be cloned by the `insert` action. If this attribute is present and resolves to a non-empty nodeset, then the result overrides the default setting of the `origin node-set` as described below in the processing of the `insert` action.

at

Author-optional attribute containing an XPath expression evaluated using the Node Set Binding node-set to help determine the `insert location node`. This attribute is ignored if the Node Set Binding is not specified or specifies an empty node-set. The `insert location node` is a node within the Node Set Binding node-set that is used to help determine where in the instance to insert each node cloned by the `insert`. If this attribute is present, then its result is used to override the default setting of the `insert location node` as described below in the processing of the `insert` action.

position

Author-optional selector that indicates where to put the cloned node or nodes relative to the `insert location node`. Valid values are `before` and `after`, and the latter is the default. This attribute is ignored if the Node Set Binding node-set is not specified or empty. If the node at the `insert location node` within the Node Set Binding node-set is the document element of an instance, then this attribute is ignored.

Processing for the `insert` action is as follows:

1. The `insert context` is determined. If the `context` attribute is not given, the `insert context` is the in-scope evaluation context. Otherwise, the XPath expression provided by the `context` attribute is evaluated using the in-scope evaluation context, and the first node rule is applied to obtain the `insert context`. The `insert` action is terminated with no effect if the `insert context` is the empty node-set.
2. The Node Set Binding node-set is determined. If a `bind` attribute is present, it directly determines the Node Set Binding node-set. If a `nodeset` attribute is present, it is evaluated within the `insert context` to determine the Node Set Binding node-set. If the Node Set Binding attributes are not present, then the Node Set Binding node-set is the empty node-set. The `insert` action is terminated with no effect if any of the following conditions is true:
 - a. The `context` attribute is not given and the Node Set Binding node-set is the empty node-set.
 - b. The `context` attribute is given, the `insert context` does not evaluate to an element node and the Node Set Binding node-set is the empty node-set.
3. The `origin node-set` is determined. If the `origin` attribute is not given and the Node Set Binding node-set is empty, then the `origin node-set` is the empty node-set. Otherwise, if the `origin` attribute is not given, then the `origin node-set` consists of the last node of the Node Set Binding node-set. If the `origin` attribute is given, the `origin node-set` is the result of the evaluation of the `origin` attribute in the `insert context`. Namespace nodes and root nodes (parents of document elements) are removed from the `origin node-set`. The `insert` action is terminated with no effect if the `origin node-set` is the empty node-set.
4. The `insert location node` is determined. If the Node Set Binding node-set is not specified or empty, the `insert location node` is the `insert context node`. Otherwise, if the `at` attribute is not given, then the `insert location node` is the last node of the Node Set Binding node-set. Otherwise, an `insert location node` is determined from the `at` attribute as follows:
 - a. The evaluation context node is the first node in document order from the Node Set Binding node-set, the context size is the size of the Node Set Binding node-set, and the context position is 1.
 - b. The return value is processed according to the rules of the XPath function `round()`. For example, the literal 1.5 becomes 2, and the literal 'string' becomes NaN.
 - c. If the result is in the range 1 to the Node Set Binding node-set size, then the `insert location` is equal to the result. If the result is non-positive, then the `insert location` is 1. Otherwise, the result is NaN or exceeds the Node Set Binding node-set size, so the `insert location` is the Node Set Binding node-set size.
 - d. The `insert location node` is the node in the Node Set Binding node-set at the position given by the `insert location`.
5. The `insert` action is terminated with no effect if the insertion will create nodes whose parent is readonly. This occurs if the `insert location node` is readonly and the Node Set Binding node-set is not specified or empty, or otherwise if the parent of the `insert location node` is readonly.
6. Each node in the `origin node-set` is cloned in the order it appears in the `origin node-set`.
7. The `target location` of each of the cloned nodes is determined as follows:
 - a. If the Node Set Binding node-set is not specified or empty, then the `insert location node` provided by the `context` attribute is intended to be the parent of the cloned node. The `target location` is dependent on the types of the cloned node and the `insert location node` as follows:
 - If the `insert location node` is not an element node or root node, then it cannot be the parent of the cloned node, so the `target location` is undefined.
 - If the `insert location node` is the root node of an instance (which is the parent of the root element), and the cloned node is an element, then the `target location` is the root element of the instance.
 - If the `insert location node` is the root node of an instance (which is the parent of the root element), and the cloned node is not

an element, then the `target location` is before the first child of the `insert location node`.

- If the `insert location node` is an element, and the cloned node is an attribute, then the `target location` is the attribute list of the `insert location node`.
 - If the `insert location node` is an element, and the cloned node is not an attribute, then the `target location` is before the first child of the `insert location node`, or the child list of the `insert location node` if it is empty.
- b. Otherwise, the Node Set Binding node-set is specified and non-empty, so the `insert location node` provided by the Node Set Binding and author-optional `at` attribute is intended to be the sibling of the cloned node. If the `insert location node` is an attribute or root node, then the `target location` is undefined. If the `insert location node` is not an attribute or root node, then the `target location` is immediately before or after the `insert location node`, based on the `position` attribute setting or its default.
8. The cloned node or nodes are inserted in the order they were cloned into their `target locations` depending on their node type. If the parent node of the target location is the instance root node (which is the parent of the root document element of the instance), and if the cloned node is an element, then the instance root element is deleted before the cloned node is inserted at the target location. If the cloned node is a duplicate of another attribute in its parent element, then either the duplicate attribute is first removed or the existing attribute value is updated. If a cloned node cannot be placed at the `target location` due to a node type conflict or because the `target location` is undefined, then the insertion for that particular cloned node is ignored. Each cloned node that is inserted is added to the `inserted-nodes` list that will be provided in the `xforms-insert` event context information. For each cloned node used to update an existing attribute node, the existing attribute node is added to the list of `inserted-nodes`.

Note:

A node type conflict is a mismatch between the XPath node type and the `target location`. For example, an attribute cannot be inserted as a sibling before or after an element.

9. If the list of `inserted-nodes` is empty, then the `insert` action is terminated with no effect.
10. The XForms action system's deferred update flags for rebuild, recalculate, revalidate and refresh are set.
11. The `insert` action is successfully completed by dispatching the `xforms-insert` event with appropriate context information.

Note:

A `repeat` updates its index in response to this event if its repeat collection changes size as a result of the insertion. See Section [9.3.3 Repeat Processing](#) for details.

Note:

This action affects [deferred updates](#) by setting the deferred update flags for rebuild, recalculate, revalidate and refresh.

Examples:

Inserting into a repeat, whether or not it is empty

When the `repeat` is empty, the `at` index is zero so a new `item` is prepended to the child elements of `purchaseOrder`. When the `repeat` is non-empty, the new `item` is added after the node currently indexed by `repeat R`.

```
...
<xforms:instance>
  <purchaseOrder xmlns="">
    <subtotal/>
    <tax/>
    <total/>
  </purchaseOrder>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    ...
    <item>
      <product/>
      <quantity/>
      <unitcost/>
      <price/>
    </item>
    ...
  </prototypes>
</xforms:instance>
...
<repeat nodeset="/purchaseOrder/item" id="R">
  ...
</repeat>
...
<xforms:trigger>
  <xforms:label>Add to purchase order</xforms:label>
  <xforms:action ev:event="DOMActivate">
    <xforms:insert context="/purchaseOrder" nodeset="item" at="index('R')" origin="instance('prototypes')/item"/>
    <xforms:setfocus control="R"/>
  </xforms:action>
</xforms:trigger>
```

Insert and Read-Only Content

```
<model xmlns:my="http://example.org">
  <instance>
    <my:data>
      <my:name>
        <my:first-name>John</my:first-name>
```

```

    <my:last-name>Doe</my:last-name>
  </my:name>
  <my:address>
    <my:street>123 Main St.</my:street>
    <my:city>Smallville</my:city>
  </my:address>
</my:data>
</instance>

<bind nodeset="/my:data/my:name/" readonly="true()" />
<bind nodeset="/my:data/my:address/my:street" readonly="true()" />

<action ev:event="xforms-model-construct-done">
  <insert id="I1" nodeset="my:name/*" ... />
  <insert id="I2" nodeset="my:address/my:street" at="1" >
</action>
</model>

```

Insert I1 fails because it attempts to insert into the content of a readonly node (`my:name`). Insert I2 succeeds even though the insert location is a readonly node because the new node is placed as a sibling into the content of the parent, which is not readonly.

See [10.4 The delete Element](#) for an example that uses `insert` and `delete` to make a `repeat` that always shows at least one repeat item. See Appendix [B Patterns for Data Mutations](#) for numerous further usage patterns for `sevalue`, `insert` and `delete`.

10.4 The delete Element

This action deletes one or more nodes from instance data.

Common Attributes: [Common](#), [Events](#), [Action Common](#), [Node Set Binding](#)

Special Attributes:

context

Author-optional attribute containing an XPath expression evaluated using the in-scope evaluation context. If the `model` attribute is present, then it is processed as described in [7.2 Evaluation Context](#) before evaluating this attribute. The Node Set Binding is required unless this attribute is present. The result of the XPath expression is used to override the in-scope evaluation context. If the result is an empty nodeset or not a nodeset, then the `delete` action is terminated with no effect. Otherwise, the first node of the nodeset is used as the new in-scope evaluation context node, and the context position and size are set to 1. By adjusting the in-scope evaluation context, this attribute affects the evaluation of subsequent attributes that may appear on `delete`, including `if`, `while`, and `nodeset`.

at

Author-optional attribute containing an XPath expression evaluated using the Node Set Binding node-set to determine the `delete location`. If the Node Set Binding node-set is empty, then this attribute is ignored.

Processing for the `delete` action is as follows:

1. The `delete context` is determined. It is set to the in-scope evaluation context, possibly overridden by the `context` attribute if that attribute is present. The `delete` action is terminated with no effect if the `delete context` is the empty node-set.
2. The Node Set Binding node-set is determined. If a `bind` attribute is present, it directly determines the Node Set Binding node-set. If a `nodeset` attribute is present, it is evaluated within the `delete context` to determine the Node Set Binding node-set. The behavior of the `delete` action is undefined if the Node Set Binding node-set contains nodes from more than one instance. The `delete` action is terminated with no effect if the Node Set Binding is expressed and the Node Set Binding node-set is the empty node-set. Otherwise, the Node Set Binding is not expressed, so the Node Set Binding node-set is set equal to the `delete context` node with a position and size of 1.
3. The `delete location` is determined. If the `at` attribute is not specified, there is no `delete location`. Otherwise, the `delete location` is determined by evaluating the XPath expression specified by the `at` attribute as follows:
 - a. The evaluation context node is the first node in document order from the Node Set Binding node-set, the context size is the size of the Node Set Binding node-set, and the context position is 1.
 - b. The return value is processed according to the rules of the XPath function `round()`. For example, the literal 1.5 becomes 2, and the literal 'string' becomes NaN.
 - c. If the result is in the range 1 to the Node Set Binding node-set size, then the `delete location` is equal to the result. If the result is non-positive, then the `delete location` is 1. Otherwise, if the result is NaN or exceeds the Node Set Binding node-set size, the `delete location` is the Node Set Binding node-set size.
4. If there is no `delete location`, each node in the Node Set Binding node-set is deleted, except if the node is a readonly node, a namespace node, a root node, or the root document element of an instance, then that particular node is not deleted. Otherwise, if there is a `delete location`, the node at the `delete location` in the Node Set Binding node-set is deleted, except if the node is the root document element of an instance or has a readonly parent node, then that node is not deleted. The `delete` action is terminated with no effect if no node is deleted.
5. The XForms action system's deferred update flags for rebuild, recalculate, revalidate and refresh are set.
6. The `delete` action is successfully completed by dispatching the `xforms-delete` event with appropriate context information.

Note:

This action affects [deferred updates](#) by setting the deferred update flags for rebuild, recalculate, revalidate and refresh.

Examples:

Using delete and insert to Maintain a Non-empty repeat repeat

In this example, the trigger is not in the repeat. When it is activated, the indexed item in the repeat is first deleted. Next, if that was the last item, then a new prototypical item is inserted so that the repeat does not become empty. The focus is then sent back to the repeat from the trigger.

```
...
<xforms:trigger>
  <xforms:label>Delete from purchase order</xforms:label>
  <xforms:action ev:event="DOMActivate">
    <xforms:delete context="/purchaseOrder" nodeset="item" at="index('R')"/>
    <xforms:insert context="/purchaseOrder" if="not(item)"
      nodeset="item" origin="instance('prototypes')/item"/>
    <xforms:setfocus control="R"/>
  </xforms:action>
</xforms:trigger>
```

Note:

The form author could have written `nodeset="/purchaseOrder/item"` in the delete action, but the `context` attribute was added for consistency with the insert action.

Delete and Read-Only Content

```
<model xmlns:my="http://example.org">
  <instance>
    <my:data>
      <my:name>
        <my:first-name>John</my:first-name>
        <my:last-name>Doe</my:last-name>
      </my:name>
      <my:address>
        <my:street>123 Main St.</my:street>
        <my:city>Smallville</my:city>
      </my:address>
    </my:data>
  </instance>

  <bind nodeset="/my:data/my:name/" readonly="true()"/>
  <bind nodeset="/my:data/my:address/my:street" readonly="true()"/>

  <action ev:event="xforms-model-construct-done">
    <delete id="D1" nodeset="my:name/*" ... />
    <delete id="D2" nodeset="my:address/my:street" at="1" >
    <delete id="D3" nodeset="my:address" at="1" >
  </action>
</model>
```

Delete D1 fails because it attempts to delete from the content of a readonly node (`my:name`). Delete D2 succeeds even though the node to delete is readonly because the node is not being changed, but rather removed from the content of the parent, which is not readonly. Delete D3 succeeds even though it contains a readonly node because a node can be deleted if its parent is not readonly, and node deletion includes deletion of its attributes and content, regardless of whether or not the attributes or content nodes are readonly.

See Appendix [B Patterns for Data Mutations](#) for numerous further usage patterns for `sevalue`, `insert` and `delete`.

10.5 The setindex Element

This XForms Action begins by invoking the [deferred update behavior](#). This action then marks a specific item as current in a repeating sequence (within [9.3.1 The repeat Element](#)).

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

repeat

Required reference to a repeating element.

index

Required XPath expression that evaluates to a 1-based offset into the sequence. The evaluation context is determined in the same manner as the evaluation context for a Single-Node Binding (see [7.2 Evaluation Context](#)).

If the selected index is 0 or less, an `xforms-scroll-first` event is dispatched and the index is set to 1. If the selected index is greater than the index of the last repeat item, an `xforms-scroll-last` event is dispatched and the index is set to that of the last item. If the index evaluates to NaN the action has no effect.

Note:

The IDREF from the `repeat` attribute may not uniquely identify the desired [repeat](#) if the `repeat` element bearing the matching ID resides within the content of another `repeat`. The general method described in [4.7 Resolving ID References in XForms](#) is used to determine the desired run-time repeat object.

Note:

This action affects [deferred updates](#) by performing deferred update in its initialization and by setting the deferred update flags for

recalculate, revalidate and refresh.

10.6 The toggle Element

This XForms Action begins by invoking the [deferred update behavior](#). This action then selects one possible case from an exclusive list of alternatives in a `switch`.

This action performs the following:

1. Dispatches an `xforms-deselect` event to the currently selected `case` in the `switch` containing the identified `case`.
2. Adjusts the `selected` states (not the attribute values) on the affected cases to reflect the new state of the `switch` containing the identified `case`.
3. Dispatches an `xform-select` event to the `case` just selected.

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

case

Author-optional reference to a `case` element. The `case` to select is specified by this attribute or by the child element `case`.

The `case` to be selected by the `switch` is identified by IDREF either by the attribute `case` or by a child `case` element. If no `case` element contains the given identifier, then this action has no effect.

Note:

Whether the IDREF is obtained from the `case` attribute or element, the IDREF may not uniquely identify the desired `case` if the `case` element bearing the matching ID resides in a repeating construct such as element `repeat`. The general method described in [4.7 Resolving ID References in XForms](#) is used to determine the desired run-time case object.

Note:

This action affects [deferred updates](#) by performing deferred update in its initialization.

10.6.1 The case Element Child of the toggle Element

This section defines a child element of `toggle` named `case` that is an alternate means of providing the identity of a `case` element to select with a `switch`.

Element: `case`

Common attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the `case` identity, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The `case` to be selected by the `toggle` action is given by the `case` attribute or the `case` element. If both are given, the element takes precedence. Due to the addition of the element, the `case` attribute is no longer required, but either the `case` attribute or the `case` element must appear. The `case` element can provide the identity of a `case` with either its string content or the `value` attribute. If both are given, then the `value` attribute takes precedence.

```
<toggle>
<case value="concat('case_', ../addressBlockType)"/>
</toggle>
```

10.7 The setfocus Element

This XForms Action begins by invoking the [deferred update behavior](#). This action then dispatches an `xforms-focus` event ([4.3.7 The xforms-focus Event](#)) to the element identified by attribute `control` or child element `control`.

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

control

Author-optional reference to a `form control`. The form control is specified by this attribute or by the child element `control`.

The identity of the element to which the `setfocus` action dispatches `xforms-focus` is given by the `control` attribute or the `control` element. If no such element contains the given identifier, then this action has no effect.

Note:

Whether the IDREF is obtained from the `control` attribute or element, the IDREF may not uniquely identify the desired form control if the element bearing the matching ID resides in a repeating construct such as element `repeat`. The general method described in [4.7 Resolving](#)

[ID References in XForms](#) is used to determine the desired form control.

Note:

Changing the focus to a form control within a repeat object may cause one or more repeat index values to be changed as described in Section [9.3.4 User Interface Interaction](#).

10.7.1 The control Element Child of the setfocus Element

This section defines a child element of `setfocus` named `control` that is an alternate means of providing the element that receives the `xforms-focus` event.

Element: `control`

Common attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the desired element identifier, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The identity of the element to which the `setfocus` action dispatches `xforms-focus` is given by the `control` attribute or the `control` element. If both are given, the element takes precedence. Due to the addition of the element, the `control` attribute is no longer required, but either the `control` attribute or the `control` element must appear. The `control` element can provide the desired element identifier with either its string content or the `value` attribute. If both are given, then the `value` attribute takes precedence.

```
<setfocus>
  <control value="concat('input_', ../paymentType)"/>
</setfocus>
```

10.8 The dispatch Element

This action dispatches an XML Event to a specific target element. Two kinds of event can be dispatched:

1. Predefined XForms events (i.e., `xforms-event-name`), in which case the `bubbles` and `cancelable` attributes are ignored and the standard semantics as defined in [4 Processing Model](#) apply.
2. An event created by the XForms author with no predefined XForms semantics and as such not handled by default by the XForms Processor.

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

name

Author-optional attribute for specifying the name of the event to dispatch.

targetid

Author-optional attribute for specifying the reference to the event target.

delay

Author-optional attribute that indicates the minimum number of milliseconds to delay dispatching of the event to the target. The default is the empty string, which indicates no delay.

bubbles

Author-optional attribute containing a boolean indicating if this event bubbles—as defined in [\[DOM2 Events\]](#). The default value is `true` for a custom event. For predefined events, this attribute has no effect.

cancelable

Author-optional attribute containing a boolean indicating if this event is cancelable—as defined in [\[DOM2 Events\]](#). The default value is `true` for a custom event. For predefined events, this attribute has no effect.

The event to be dispatched is given by the `name` attribute or the `name` child element. Due to the addition of the `name` element, the `name` attribute is not required, but either the `name` attribute or the `name` element must appear. If the event name is not specified or empty string, then this action has no effect.

The element to which the event is to be dispatched is identified by the `targetid` attribute or the `targetid` child element. Due to the addition of the `targetid` element, the `targetid` attribute is not required, but this action has no effect unless the target identifier is specified by the element or attribute. For backwards compatibility with documents created for earlier versions of the specification, the processor of the `dispatch` element [may](#) allow the attribute named `target` and the child element `target` to be used. The attribute and element named `target` provide exactly the same behaviors as the `targetid` attribute and element, except that the `target` attribute and element are ignored if the `dispatch` element also bears a `targetid` attribute or contains a `targetid` child element.

Note:

Whether the IDREF is obtained from the `targetid` attribute or `targetid` element, the IDREF may not uniquely identify the desired target object if the element bearing the matching ID resides in a repeating construct such as element `repeat`. The general method described in [4.7 Resolving ID References in XForms](#) is used to determine the desired target object.

The event may be dispatched immediately or after a specified non-negative number of milliseconds of delay. The event delay is specified the `delay` attribute or by the child element `delay`. If the delay is not specified or if the given value does not conform to `xsd:nonNegativeInteger`, then the event is dispatched immediately as the result of the `dispatch` action. Otherwise, the specified event is added to the delayed event queue unless an event with the same name and target element already exists on the delayed event queue. The `dispatch` action has no effect if the event delay is a non-negative integer and the specified event is already in the delayed event queue.

Note:

Since an element bearing a particular ID may be repeated, the delayed event queue may contain more than one event with the same name and target IDREF. It is the name and the target **run-time element** that must be unique.

If a run-time element is destroyed, then any delayed events targeted at that element are removed from the delayed event queue. A run-time element may be destroyed for a number of reasons, including shutdown of the form or removal of form controls associated by a `repeat` with an instance data node that is destroyed.

As soon as possible after the specified delay in milliseconds has elapsed, the event is removed from the delayed event queue and then dispatched. In the same manner used to handle user-generated events or the completion of an asynchronous submission, the dispatch and processing of delayed events is done without interrupting the processing of another event and its event handlers.

Note:

Because the delayed event is first removed from the delayed event queue and then dispatched, a handler for a given event may dispatch the event again with a delay. This can be used to perform simple polling and asynchronous looping operations. Moreover, the `if` attribute can be applied to the `dispatch` action to decide when to discontinue the polling or looping based on a setting in instance data.

10.8.1 The name Child Element

This section defines a new child element of `dispatch` that provides an alternate means of specifying the name of the event to dispatch.

Element: `name`

Common attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the event name, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The event name of the `dispatch` action is given by the `name` attribute or the `name` element. If both are given, the element takes precedence. The `name` element can provide the event name with either its string content or the `value` attribute. If both are given, then the `value` attribute takes precedence.

10.8.2 The targetid Child Element

This section defines a new child element of `dispatch` that provides an alternate means of specifying the target of the event to be dispatched.

Element: `targetid`

Common attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the event target, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The event target of the `dispatch` action is given by the `targetid` attribute or the `targetid` element. If both are given, the element takes precedence. The `targetid` element can provide an IDREF for the event target with either its string content or the `value` attribute. If both are given, then the `value` attribute takes precedence.

10.8.3 The delay Child Element

This section defines a new child element of `dispatch` that provides an alternate means of specifying the delay imposed on the event to be dispatched.

Element: `delay`

Common attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the event delay, the

result of the expression is processed as if by call to the XPath `string` function. If the result does not conform lexically to `xsd:nonNegativeInteger`, then the result of empty string is used. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The event delay of the `dispatch` action is given by the `delay` attribute or the `delay` element. If both are given, the element takes precedence. The `delay` element can provide the delay with either its string content or the `value` attribute. If both are given, then the `value` attribute takes precedence.

10.9 The rebuild Element

This action causes the default processing of `xforms-rebuild` to happen, bypassing the normal event flow (i.e. the behavior occurs without dispatching the `xforms-rebuild` event). This action results in the XForms Processor rebuilding any internal data structures used to track computational dependencies among instance data nodes—see [4.3.1 The xforms-rebuild Event](#).

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

model

Author-optional XForms Model selector, as defined in [3.2.3 Single-Node Binding Attributes](#). If this attribute is omitted, then the default is the `model` associated with the in-scope evaluation context node.

Note:

This action affects [deferred updates](#).

10.10 The recalculate Element

This action causes the default processing of `xforms-recalculate` to happen, bypassing the normal event flow (i.e. the behavior occurs without dispatching the `xforms-recalculate` event). As a result, instance data nodes whose values need to be recalculated are updated as specified in the processing model—see [4.3.2 The xforms-recalculate Event](#).

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

model

Author-optional XForms Model selector, as defined in [3.2.3 Single-Node Binding Attributes](#). If this attribute is omitted, then the default is the `model` associated with the in-scope evaluation context node.

Note:

This action affects [deferred updates](#).

10.11 The revalidate Element

This action causes the default processing of `xforms-revalidate` to happen, bypassing the normal event flow (i.e. the behavior occurs without dispatching the `xforms-revalidate` event). This results in the instance data being revalidated as specified by the processing model—see [4.3.3 The xforms-revalidate Event](#).

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

model

Author-optional XForms Model selector, as defined in [3.2.3 Single-Node Binding Attributes](#). If this attribute is omitted, then the default is the `model` associated with the in-scope evaluation context node.

Note:

This action affects [deferred updates](#).

10.12 The refresh Element

This action causes the default processing of `xforms-refresh` to happen, bypassing the normal event flow (i.e. the behavior occurs without dispatching the `xforms-refresh` event). This action results in the XForms user interface being *refreshed*, and the presentation of user interface controls being updated to reflect the state of the underlying instance data—see [4.3.4 The xforms-refresh Event](#).

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

model

Author-optional XForms Model selector, as defined in [3.2.3 Single-Node Binding Attributes](#). If this attribute is omitted, then the default is the `model` associated with the in-scope evaluation context node.

Note:

This action affects [deferred updates](#).

10.13 The reset Element

This action initiates reset processing by dispatching an `xforms-reset` event to the specified `model`. Processing of event `xforms-reset` is defined in the processing model—see [4.3.5 The xforms-reset Event](#).

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

model

Author-optional XForms Model selector, as defined in [3.2.3 Single-Node Binding Attributes](#). If this attribute is omitted, then the default is the `model` associated with the in-scope evaluation context node.

Note:

This action affects [deferred updates](#).

10.14 The load Element

This action traverses the specified link.

Common Attributes: [Common](#), [Events](#), [Action Common](#), [Single-Node Binding](#) (author-optional)

Special Attributes:

resource

Author-optional attribute. Link to an external resource to load.

show

Author-optional link behavior specifier. The allowed values are "replace" and "new". If this attribute is missing, a default value of "replace" is assumed.

The URI specifying the link to traverse may be pointed to by the Single Node Binding attributes, if given, or by the `resource` attribute or the `name` child element. Individually, the Single Node Binding, `resource` element and `resource` attribute are not required. If none are given, the action has no effect. If the Single Node Binding is present and does not select an instance data node, then this action has no effect. If the Single Node Binding is given in addition to one of the `resource` attribute or `resource` element, then the action has no effect.

The URI obtained in this manner is treated as a link to an external resource, defined as an [XLink 1.0](#) link between the `load` element and the remote resource indicated. No XLink `actuate` value is defined, since control of actuation is defined by XML Events. The XLink `show` value depends on the `show` attribute.

The link indicated by the URI obtained above is traversed. If the link traversal fails, then an implementation-specific means of conveying the link traversal failure occurs. Otherwise, processing for the document (or portion of the document) reached by traversing the link is specified by the `show` attribute. The following are the possible values for the `show` attribute and the corresponding processing behaviors:

new

The document is loaded into a new presentation context, e.g., a new window. Form processing in the original window continues.

replace

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

10.14.1 The resource Element child of load

When it appears as the first child element of `load`, the `resource` element provides the URI of the link, overriding the `resource` attribute. As stated above, the `load` action has no effect if both a `resource` and a Single Node Binding are given. This element allows the URI used by the `load` to be dynamically calculated based on instance data.

Common Attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the URI, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The URI to be used by the `load` can be specified with either the `value` attribute or the string content of the `resource` element. If both are specified, then the `value` attribute takes precedence. If the `load` does not have a `resource` element as its first child, then the URI is obtained from the `resource` attribute or the Single Node Binding, if given.

10.15 The send Element

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

Common Attributes: [Common](#), [Events](#), [Action Common](#)

Special Attributes:

submission

Author-optional attribute containing a reference to a `submission` element. If this attribute is given but does not identify a `submission` element, then the `send` action has no effect. If this attribute is omitted, then the first `submission` in document order from the `model` associated with the in-scope evaluation context is used.

10.16 The message Element

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

Common Attributes: [Common](#), [Events](#), [Action Common](#), [Single Node Binding](#) (author-optional)

Special Attributes:

level

Author-optional attribute containing a message level identifier, one of ("ephemeral"|"modeless"|"modal"|[QNameButNotNCName](#)). The default is "modal" if the attribute is not specified. This specification does not define behavior for [QNameButNotNCName](#) values.

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

The user interface for the `message` action is considered to be created at the time the action occurs. If the message is obtained from the inline content of the `message` action, then the output of any `output` controls in the `message` content is determined based on the instance data available when the `message` action occurs. For example, the following example displays the message `Hello, world!` as the form starts up:

```
<model>
  <instance>
    <data xmlns="">
      <name>John</name>
    </data>
  </instance>

  <action ev:event="xforms-ready">
    <setvalue ref="name">world</setvalue>
    <message level="modal">Hello, <output ref="name"/>!</message>
  ...</action>
</model>
```

In this example, the message includes the latest user input even though other form controls not in the `message` action are not guaranteed to be updated until the end of the `xforms-refresh` event processing:

```
<input ref="birthday">
  <label>Enter birthday:</label>
  <message ev:event="xforms-invalid"><output ref="."/> isn't a valid birthday</message>
</input>
```

Note:

Due to deferred update behavior, if a `message` action is preceded in an action sequence by other actions that change instance nodes, and the message references nodes that are computationally dependent on the changed nodes, then the form author should invoke the `recalculate` action before the `message` action. Moreover, if the computational dependencies involved nodes that were inserted or deleted, then the form author should invoke `rebuild` prior to the `recalculate`.

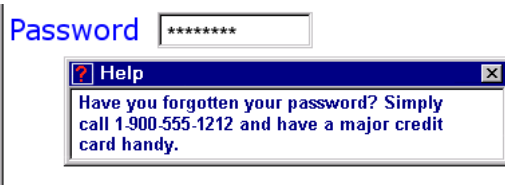
A graphical browser might render a modal message as follows:

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



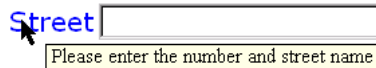
A modeless message is the foundation for displaying a `help` message, which a graphical browser might render as follows:

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



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

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



10.17 Conditional Execution of XForms Actions

The `if` attribute can be added to any XForms action. It contains an [XPath 1.0](#) expression that is evaluated using the in-scope evaluation context before the action is executed. The result of the expression is converted to a `boolean` as if converted with the `boolean()` function defined by the [XPath 1.0](#) specification. If the converted result of the expression evaluates to `false`, then the action is not performed. If the converted result is `true`, then the action is performed.

If this attribute is applied to an XForms `action` element and the converted result of evaluation is `false`, then all of the actions within the `action` element are omitted from the execution of the XForms action sequence that invoked the `action` element. If the result is `true`, then the contained actions are performed according to the normal processing rules such as deferred update behavior and applicability of conditional and iterative attributes.

Note:

In actions `insert` and `delete`, the attribute `context` is evaluated before the `if` attribute.

Automatic Focus Advancement

The `setfocus` action in each input control is executed only if the node bound to the control is a number of a particular length. The exacting form author could perform further validity tests.

```
...
<input ref="areaCode" id="AreaCodeControl" incremental="true">
  <label>Area Code</label>
  <setfocus ev:event="xforms-value-changed" control="ExchangeControl" if="string-length(.)=3 and . > 0"/>
</input>
<input ref="exchange" id="ExchangeControl" incremental="true">
  <label>Exchange</label>
  <setfocus ev:event="xforms-value-changed" control="LocalControl" if="string-length(.)=3 and . > 0"/>
</input>
<input ref="local" id="LocalControl" incremental="true">
  <label>Local</label>
  <setfocus ev:event="xforms-value-changed" control="ExtensionControl" if="string-length(.)=4 and . > 0"/>
</input>
...
```

Handling Focus for Empty Repeats

The trigger that performs a delete conditionally sets the focus to a control outside of the repeat if the repeat becomes empty due to the deletion. The `setfocus` is called first because the `delete` removes the context node.

```
...
<trigger id="InsertControl">
  <label>Insert Row</label>
  <action ev:event="DOMActivate">
    <insert context="purchaseOrder/lines" nodeset="line"
      at="index('PurchaseOrderRepeat')" origin="instance('prototype')"/>
    <setfocus control="PurchaseOrderRepeat"/>
  </action>
</trigger>
<repeat nodeset="purchaseOrder/lines/line" id="PurchaseOrderRepeat">
  ...
  <trigger>
    <label>Delete Row</label>
    <action ev:event="DOMActivate">
      <setfocus control="InsertControl" if="last()=1"/>
      <delete nodeset="../line" at="index('PurchaseOrderRepeat')"/>
    </action>
  </trigger>
  ...
</repeat>
```

10.18 Iteration of XForms Actions

The `while` attribute can be added to any XForms action. It contains an [XPath 1.0](#) expression that is evaluated using the in-scope evaluation context before the action is executed. The result of the expression is converted to a `boolean` as if converted with the `boolean()` function defined by the [XPath 1.0](#) specification. If the converted result of the expression is `true`, then the XForms action is performed and then the expression is re-evaluated. The XForms action is executed repeatedly until the converted result of the expression evaluates to `false`.

If this attribute is applied to an XForms `action` element, then the sequence of XForms actions in its content are executed repeatedly once for each time the immediately preceding evaluation of the expression yields a result of `true`.

When XForms actions are iteratively executed, they are still subject to the normal action processing rules such as deferred update and applicability of conditional and iterative attributes.

An XForms action may be executed zero times due to this attribute. Furthermore, if an action bears this attribute and the `if` attribute, then the expressions of both attributes must evaluate to `true` before each iterative execution of the action.

Note:

In actions `insert` and `delete`, the attribute `context` is evaluated before the `while` attribute. Therefore, `context` is re-evaluated before each iteration of the actions controlled by the `while` attribute.

Summing Selected Results

Counter and accumulator variables are created in instance data to sum a selection of values chosen by the user

```
<trigger>
  <label>Get Sum</label>
  <action ev:event="DOMActivate">
    <setvalue ref="instance('temps')/counter" value="1"/>
    <setvalue ref="instance('temps')/accumulator" value="0"/>
    <action while="instance('temps')/counter <= count(/some/nodes)">
      <setvalue ref="instance('temps')/accumulator"
        value=". + instance('default')/some/nodes[number(instance('temps')/counter)]"
        if="boolean-from-string(/some/nodes[number(instance('temps')/counter)]/@selected)"/>
      <setvalue ref="instance('temps')/counter" value=". + 1"/>
    </action>
  </action>
</trigger>
```

10.19 Actions from Other Modules

Of the action handlers detailed in this chapter, XForms defines some to be part of the XForms Switch and Repeat modules: [10.6 The toggle Element](#) and [10.5 The setindex Element](#).

11 The XForms Submission Module

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

11.1 The submission Element

The `submission` element represents declarative instructions on what to submit, and how.

Element	Overview of Attributes	Overview of Content Model
submission	Common ref (binding-expression) bind (xsd:IDREF) resource (xsd:anyURI) action (xsd:anyURI) [deprecated] mode ("asynchronous" "synchronous") method ("post" "get" "put" "delete" "multipart-post" "form-data-post" "urlencoded-post" Any other NCName QNameButNotNCName) validate (xsd:boolean) relevant (xsd:boolean) serialization ("application/xml" "application/x-www-form-urlencoded" "multipart/related" "multipart/form-data" "none") version (xsd:NMTOKEN) indent (xsd:boolean) mediatype (xsd:string) encoding (xsd:string) omit-xml-declaration (xsd:boolean) standalone (xsd:boolean) cdata-section-elements (QNameList) replace ("all" "instance" "text" "none" QNameButNotNCName) instance (xsd:IDREF) targetref (nodeset XPath Expression) separator (';' '&') includenamespaceprefixes (xsd:NMTOKENS)	(resource method header)* , Action*

Below is a more detailed description of each attribute whose name and datatype information appears above.

Common Attributes: [Common](#)

Special Attributes:

resource

Attribute indicating the destination URI for submitting instance data. This attribute is not author-optional unless the destination URI is provided by the `resource` element, which can dynamically specify the URI based on instance data, or the `action` attribute. This attribute should be used in place of the `action` attribute. Behavior of relative URIs in links is determined by the host language, although [XML Base](#) processing is strongly recommended.

action

Deprecated author-optional attribute indicating the destination URI for submitting instance data. Behavior of relative URIs in links is determined by the host language, although [\[XML Base\]](#) processing is strongly recommended. Due to the addition of the `resource` attribute, this attribute is deprecated and optional. However, the destination URI must be specified by this attribute or by either the `resource` attribute or the `resource` element.

ref

Author-optional selector [binding expression](#) enabling submission of a portion of the instance data. The selected node, and all descendants, are selected for submission. The default value is `"/`.

bind

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

mode

Author-optional attribute defaulting to "asynchronous" and with legal values of "synchronous" and "asynchronous". This attribute controls whether or not the submission response processing is performed as part of the default processing of event `xforms-submit`. An asynchronous submission can complete default processing for this event before the submission response is received; in this case, submission response is processed once it is completely received asynchronously. For a "synchronous" submission, the submission response is received and processed during the default processing of event `xforms-submit`.

method

Author-optional attribute specifying the protocol operation to be used to transmit the serialized instance data. There is no default value because either the attribute `method` or the element `method` must be specified. See Section [11.9 Submission Options](#) for information on how this attribute affects the default serialization of instance data and the HTTP method [\[RFC 2616\]](#).

validate

Author-optional boolean attribute that indicates whether or not the data validation checks of the submission are performed. The default value is "false" if the value of `serialization` is "none" and "true" otherwise.

relevant

Author-optional boolean attribute that indicates whether or not the relevance pruning of the submission is performed. The default value is "false" if the value of `serialization` is "none" and "true" otherwise.

serialization

Author-optional attribute that controls how and whether to serialize instance data as part of the submission. The default value of this attribute is based on the submission `method` as described in Section [11.9 Submission Options](#). If the attribute value is "none", then instance data is not serialized as part of the submission. This can be useful for requests that either require no data or that have the data already gathered in the URI.

Note:

Setting `serialization` to "none" will also have the default effect of preventing relevance pruning and validation. However, the author is free to override this by setting `relevant` and/or `validate` attributes to "true".

version

Author-optional attribute specifying the `version` of XML to be serialized. The default is "1.0".

indent

Author-optional attribute specifying whether the serializer should add extra white space nodes for readability. The default is "false".

mediatype

Author-optional attribute specifying the mediatype for XML serialization of instance data. Authors should ensure that the type specified is compatible with the data being submitted, such as `application/xml` for posted XML data. The default is "application/xml".

Note:

This attribute does not affect serialization and cannot be used to override the `serialization` attribute. It is only used to provide additional information about the submission data serialization when the `serialization` is `application/xml`. For example, this attribute could be used to indicate that the content type of the XML serialization is `text/xml`.

encoding

Author-optional attribute specifying an encoding for serialization. The default is "UTF-8".

omit-xml-declaration

Author-optional attribute specifying whether to omit the XML declaration on the serialized instance data. The default value is "false".

standalone

Author-optional attribute specifying whether to include a standalone declaration in the serialized XML. If the `omit-xml-declaration` attribute has the value `true`, then this attribute is ignored. Otherwise, if this attribute is omitted, then the XML declaration does not include a standalone document declaration, and if this attribute is specified, then the XML declaration includes a standalone document declaration with the same value as this attribute.

cdata-section-elements

Author-optional attribute specifying element names to be serialized with CDATA sections. The default is empty string.

replace

Author-optional attribute specifying how the information returned after submit should be applied. In the absence of this attribute, "all" is assumed. The legal values are "all", "instance", "text" and "none".

instance

Author-optional attribute specifying the instance to replace when the `replace` attribute value is "instance". When the attribute is absent, then the default is the instance that contains the submission data. An `xforms-binding-exception` ([4.5.1 The xforms-binding-exception Event](#)) occurs if this attribute does not indicate an instance in the same model as the submission.

targetref

Author-optional attribute containing an XPath expression that indicates the target node for data replacement. The in-scope evaluation context of the `submission` element is used to evaluate the expression. If `replace` is "instance", then the target node is replaced by the submission result. If `replace` is "text", then the content of the target node is replaced by the submission result. For other values of the `replace` attribute, this attribute is ignored. By default, the target node is the document element of the instance indicated by the `instance` attribute.

separator

Author-optional attribute specifying the separator character between name/value pairs in urlencoding. The default value is '&'. To express the default, character entity encoding is used: `separator='&'`.

includenamespaceprefixes

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

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

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

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

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

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

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

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

Note:

The following XSLT attributes have no counterpart in XForms:

- doctype-system
- doctype-public

The table below provides an overview of the child elements of the XForms `submission` element, including their attributes and content models. Elements defined in the XForms Actions module are also allowed in the content model of `submission`.

Element	Overview of Attributes	Overview of Content Model
resource	value (string XPath Expression)	PCDATA
method	value (string XPath Expression)	PCDATA
header	nodeset (nodeset XPath Expression)	name, value+
Action	various	various

11.2 The xforms-submit Event

Target: `submission`

Bubbles: Yes

Cancelable: Yes

Context Info: None

Under no circumstances can more than a single concurrent submit process be under way for a particular XForms submission. From the start of the default action of `xforms-submit` for a submission, until immediately before `xforms-submit-done` or `xforms-submit-error` is dispatched to that submission, the default action for subsequent `xforms-submit` events dispatched to that submission is to dispatch `xforms-submit-error` to that submission with context information containing an `error-type` of `submission-in-progress`.

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

1. The data model is updated based on some of the flags defined for [deferred updates](#). Specifically, if the deferred update `rebuild` flag is set for the `model` containing this submission, then the rebuild operation is performed without dispatching an event to invoke the operation. Then, if the deferred update `recalculate` flag is set for the `model` containing this submission, then the recalculate operation is performed without dispatching an event to invoke the operation. This sequence of operations affects the [deferred update behavior](#) by clearing the deferred update flags associated with the operations performed.
2. If the binding attributes of `submission` indicate an empty nodeset or a node other than an element or an instance document root node, then submission processing is stopped after dispatching event `xforms-submit-error` with context information containing an `error-type` of `no-data`. Otherwise, the binding attributes of `submission` indicate a node of instance data.
3. The indicated node and all nodes for which it is an ancestor are selected. If the attribute `relevant` is `true`, whether by default or declaration, then any selected node which is not relevant as defined in [6.1.4 The relevant Property](#) is deselected (pruned). If all instance nodes are deselected, then submission processing is stopped after dispatching event `xforms-submit-error` with context information containing an `error-type` of `no-data`.
4. If the attribute `validate` is `true`, whether by default or declaration, then all selected instance data nodes are checked for validity according to the definition in [4.3.3 The xforms-revalidate Event](#) (no notification events are marked for dispatching due to this operation). Any selected instance data node that is found to be invalid stops submission processing after dispatching event `xforms-submit-error` with context information containing an `error-type` of `validation-error`.
5. The [submission method](#) is determined.
6. The [submission resource](#) is determined. If the resource is not specified, then submission processing stops after dispatching even `xforms-submit-error` with context information containing an `error-type` of `resource-error`.
7. If the `serialization` attribute value is `"none"`, then the submission data serialization is the empty string. Otherwise, the submission data serialization is determined as follows. The event `xforms-submit-serialize` is dispatched. If the `submission-body` property of the event is changed from the initial value of empty string, then the content of the `submission-body` property string is used as the submission data serialization. Otherwise, the submission data serialization consists of a serialization of the selected instance data according to the rules stated in [11.9 Submission Options](#).
8. The `submission` headers are determined using the header entries produced by the `header` element(s) in the submission and the `mediatype` attribute or its default.
9. The submission is performed based on the `submission` headers, [submission method](#), [submission resource](#), and submission data serialization. The exact rules of submission are based on the URI scheme and the [submission method](#), as defined in [11.9 Submission Options](#).

Note:

A submission with no resource specification can be used to test validity of data. If the selected data is invalid, then the `xforms-submit-error` has an `error-type` of `validation-error`. If the selected data is valid, then the `xforms-submit-error` has an `error-type` of `resource-error`.

If the `mode` of the submission is `asynchronous`, then default processing for this event ends after the above steps, and submission processing is resumed once the response from the submission is returned. In the same manner used to handle user-generated events or the dispatch and processing of delayed events, the processing of the asynchronous submission response is done without interrupting the processing of any other event and its event handlers. If the `mode` of the submission is `synchronous`, then the XForms processor suspends user interaction with all form controls of the document and action processing is blocked within the default processing for this event until the response from the submission is returned.

The response returned from the submission is applied as follows:

- For a success response including a body, when the value of the `replace` attribute on element `submission` is `"all"`, the event `xforms-submit-done` may be dispatched with appropriate context information, and submit processing concludes with entire containing document being replaced with the returned body.
- For a success response including a body of an XML media type (as defined by the content type specifiers in [RFC 3023](#)), when the value of the `replace` attribute on element `submission` is `"instance"`, the response is parsed as XML. If the parse fails, then submission processing concludes after dispatching `xforms-submit-error` with appropriate context information, including an `error-type` of `parse-error`. However, if the XML parse succeeds, then instance data replacement is performed according to the rules specified in [11.10 Replacing Data with the Submission Response](#). This operation may fail for a number of reasons, including if processing of the `targetref` attribute yields a readonly node (if `replace="text"`) or a node that either has a readonly parent or is not an element (if `replace="instance"`). In this case, submission ends after dispatching event `xforms-submit-error` with appropriate context information, including an `error-type` of `target-error`. Otherwise, the instance data replacement succeeds. Submission processing then concludes after dispatching `xforms-submit-done` with appropriate context information.
- For a success response including a body of a non-XML media type (i.e. with a content type not matching any of the specifiers in [RFC 3023](#)), when the value of the `replace` attribute on element `submission` is `"instance"`, nothing in the document is replaced and submission processing concludes after dispatching `xforms-submit-error` with appropriate context information, including an `error-type` of `resource-error`.
- For a success response including a body of an XML media type (as defined by the content type specifiers in [RFC 3023](#)) or a text media type (as defined by a content type of `text/*`), when the value of the `replace` attribute on element `submission` is `"text"`, the response is

encoded as text. Then, the content replacement is performed according to the rules specified in [11.10 Replacing Data with the Submission Response](#). If the processing of the `targetref` attribute (including its default) fails, then the submission processing concludes after dispatching `xforms-submit-error` with appropriate context information, including an `error-type` of `target-error`. Otherwise, submission processing then concludes after dispatching `xforms-submit-done` with appropriate context information.

- For a success response including a body that is both a non-XML media type (i.e. with a content type not matching any of the specifiers in [\[RFC 3023\]](#)) and a non-text type (i.e. with a content type not matching `text/*`), when the value of the `replace` attribute on element `submission` is `"text"`, nothing in the document is replaced and submission processing concludes after dispatching `xforms-submit-error` with appropriate context information, including an `error-type` of `resource-error`.
- For a success response including a body, when the value of the `replace` attribute on element `submission` is `"none"`, submission processing concludes after dispatching `xforms-submit-done` with appropriate context information.
- For a success response not including a body, submission processing concludes after dispatching `xforms-submit-done` with appropriate context information.
- Behaviors of other possible values for attribute `replace` are not defined in this specification.
- For an error response, when the value of the `replace` attribute on element `submission` is `"all"`, either the document is replaced with an implementation-specific indication of an error or submission processing concludes after dispatching `xforms-submit-error` with appropriate context information, including an `error-type` of `resource-error`.
- For an error response, when the value of the `replace` attribute on element `submission` is not `"all"`, nothing in the document is replaced, and submission processing concludes after dispatching `xforms-submit-error` with appropriate context information, including an `error-type` of `resource-error`.

In addition to initiating a submission with its default processing, XForms actions can also provide handlers for the `xforms-submit` event to perform tasks such as data preparation.

Example:

Preparing data for submission

```
<submission resource="http://example.com/searchDoctors" method="post" ref="instance('doctorSearchParams') "
  replace="instance" targetref="instance('doctorList') " >
  <action ev:event="xforms-submit">
    <setvalue ref="diagnosis" value="instance('patientRecord')/malady"/>
    <setvalue ref="city" value="instance('patientRecord')/city"/>
  </action>
</submission>
```

The schema for the doctor search service requires certain portions of the patient record in order to provide a list of specialists who could treat the patient. The server-side module may perform database searches for doctors with the required specialties as well as implement business rules such as providing doctors that are within an acceptable distance of the given city. The resulting list is provided to a separate instance so it can be presented to the user for selection or used in subsequent availability searches.

11.3 The xforms-submit-serialize Event

Dispatched at the beginning of `submission` serialization (see [11.2 The xforms-submit Event](#)).

Target: `submission`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
<code>submission-body</code>	<code>node-set</code>	A document element node with a QName of <code>submission-body</code> . The node initially contains an empty string. Event handlers can write data into the node. If the string value of this node is non-empty, then the string value is used in the submission in lieu of the default instance data serialization.

Note:

The `submission-body` property is a string, but the `event()` function encapsulates the string in a text node so that the string can be modified by the `setvalue` action, which sets a value into a node determined by its Single Node Binding.

Note:

Since the `submission-body` is a string, this feature may be used to submit non-XML data.

Default Action: If the event context `submission-body` property string is empty, then no operation is performed so that the `submission` will use the normal serialization data (see [11.2 The xforms-submit Event](#)). Otherwise, if the event context `submission-body` property string is non-empty, then the serialization data for the `submission` is set to be the content of the `submission-body` string.

Example:

Submitting plain text

```
<submission resource="http://example.com/greeter" method="post" mediatype="text/plain">
  <setvalue ev:event="xforms-submit-serialize" ref="event('submission-body') " value="my/text"/>
</submission>
```

The string value of the element `my/text` is placed into the node representing the submission body, so that is the text posted by the submission.

In this example, the result returned by the submission replaces the document. This feature could be used to submit plain text, but it could also be used to allow a document to submit its serialization rather than just submitting instance data.

11.4 The xforms-submit-done Event

Dispatched as an indication of: successful completion of a submission process

Target: `submission`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
resource-uri	string	The submission resource URI that succeeded (xsd:anyURI)
response-status-code	number	The protocol return code of the success response, or NaN if the submission did not receive a success response.
response-headers	node-set	Zero or more elements, each one representing a content header in the success response received by the submission. The returned node-set is empty if the submission did not receive a response or if there were no headers. Each element has a local name of <code>header</code> with no namespace URI and two child elements, <code>name</code> and <code>value</code> , whose string contents are the name and value of the header, respectively.
response-reason-phrase	string	The protocol response reason phrase of the success response. The string is empty if the submission did not receive a response or if the response did not contain a reason phrase.

Default Action: None; notification event only.

Example:

Submission Sequencing

```
<submission resource="https://example.com/getRecord" method="post" replace="instance" instance="record">
  <send ev:event="xforms-submit-done" submission="chargeForRecord"/>
</submission>
<submission id="chargeForRecord" resource="https://example.com/chargeForRecord" method="get" serialization="none" replace="":
```

The default instance data is submitted as the search criteria for a desired record. Only upon successful completion of the submission is a second submission performed to charge the user's account for the record.

11.5 The xforms-submit-error Event

Dispatched as an indication of: failure of a submission process

Target: `submission`

Bubbles: Yes

Cancelable: No

Context Info:

Property	Type	Value
error-type	string	One of the following: <code>submission-in-progress</code> , <code>no-data</code> , <code>validation-error</code> , <code>parse-error</code> , <code>resource-error</code> , <code>target-error</code> .
resource-uri	string	The submission resource URI that failed (xsd:anyURI)
response-status-code	number	The protocol return code of the error response, or NaN if the failed submission did not receive an error response.
response-headers	node-set	Zero or more elements, each one representing a content header in the error response received by a failed submission. The returned node-set is empty if the failed submission did not receive an error response or if there were no headers. Each element has a local name of <code>header</code> with no namespace URI and two child elements, <code>name</code> and <code>value</code> , whose string contents are the name and value of the header, respectively.
response-reason-phrase	string	The protocol response reason phrase of the error response. The string is empty if the failed submission did not receive an error response or if the error response did not contain a reason phrase.
response-body	object (string or node-set)	When the error response specifies an XML media type as defined by RFC 3023 , the response body is parsed into an XML document and the root element of the document is returned. If the parse fails, or if the error response specifies a text media type (starting with <code>text/</code>), then the response body is returned as a string. Otherwise, an empty string is returned.

Default Action: None; notification event only.

Example:

Reporting a Submission Error

```
<submission resource="https://example.com/getRecord" method="post" replace="instance" instance="record">
  <message ev:event="xforms-submit-error">A submission error (<output value="event('error-type')"/>) occurred.</message>
</submission>
```

The default instance data is submitted as the search criteria for a desired record. Only upon successful completion of the submission is a second submission performed to charge the user's account for the record.

11.6 The Submission Resource

The `submission resource` is the URI for the submission. It is of type `xsd:anyURI`.

In XForms 1.0, the URI for submission was provided by the `action` attribute. For consistency, form authors should now use the attribute `resource` of type `xsd:anyURI`, which deprecates the `action` attribute. If both `action` and `resource` are present, then the `resource` attribute takes precedence.

The `resource` element provides the submission URI, overriding the `resource` attribute and the `action` attribute. If a `submission` has more than one `resource` child element, the first `resource` element child must be selected for use. Individually, the `resource` element, the `resource` attribute and the `action` attribute are not required. However, one of the three is mandatory as there is no default submission resource.

11.6.1 The resource Element

The `resource` element allows the URI used for a submission to be dynamically calculated based on instance data.

Common Attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the URI, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The URI to be used by the `submission` can be specified with either the `value` attribute or the string content of the `resource` element. If both are specified, then the `value` attribute takes precedence. If the `submission` does not have a `resource` child element, then the submission URI is obtained from the `resource` attribute or the `action` attribute.

Example:

Submitting the default instance to a location determined dynamically from an instance

```
<submission method="post">
  <resource value="instance('params')/anyURI"/>
</submission>
```

11.7 The Submission Method

The `submission method` indicates the submission protocol operation to be performed.

The submission method may be specified by the `method` attribute. The `submission` element can have a child element named `method`, which overrides the submission method setting obtained from the `method` attribute if both are specified. If more than one `method` element is given, the first occurrence in document order must be selected for use. Individually, the `method` element and the `method` attribute are not required. However, one of the two is mandatory as there is no default submission method.

11.7.1 The method Element

The `method` element allows the submission method to be dynamically calculated based on instance data.

Common Attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the method, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

Content: PCDATA

The method to be used by the `submission` can be specified with either the `value` attribute or the string content of the `method` element. If both are specified, then the `value` attribute takes precedence. If the `submission` does not have a `method` child element, then the submission method is obtained from the `method` attribute.

11.8 The header Element

The `header` element can be used to contribute information to the preamble of a submission in a manner appropriate to the protocol. The `submission` element can contain zero or more `header` child elements. Each produces zero or more header entries containing a name, a value, and a combination. The entries are provided to the submission protocol in the specified order. It is the responsibility of the submission protocol implementation to combine the entries and to serialize the result into submission protocol headers. Accordingly, entries may be re-ordered, combined, or otherwise altered in accordance with the specific protocol implementation requirements.

Common Attributes: None

Special Attributes:

nodeset

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. One or more header entries are generated for each node selected by this attribute.

combine

Author-optional attribute defaulting to "append" and with legal values of "append", "prepend", and "replace". This attribute controls the method of combination for entries produced by this element `header` with other entries produced by other `header` elements. This attribute and its default also provide information for the protocol implementation, which may use some or all of the information to combine XForms submission headers with headers provided by the user agent.

Content: (name, value+) | (value+, name)

If the `header` element does not contain a `nodeset` attribute, then one header entry is created for each `value` element. If the `header` element contains a `nodeset` attribute, then for each selected node, one header entry is created for each `value` element. The name and value of the header entry are obtained from the required child elements `name` ([11.8.1 The name Element](#)) and `value` ([11.8.2 The value Element](#)). If the name obtained from the `name` element is the empty string, then the header entry is omitted.

The header entry order is determined as follows:

1. document order of `header` elements
2. node order of nodes in `nodeset` attribute
3. document order of `value` elements

The application of this order information to header serialization is determined by the submission protocol.

If a `header` element defines the `Content-type` header, then this setting overrides a `Content-type` set by the `mediatype` attribute.

In the case of a multipart submission, the header entries are combined with those for the first part of the submission.

Example:

Setting the Accept header

In the example below, the submission request uses the `header` element to replace the user agent's existing value of the HTTP Accept header with `application/sparql-results+xml`.

```
<submission id="loadConcepts"
  resource="http://example.com/taxonomy/concepts" method="get"
  ref="instance('conceptsList') " replace="instance">
  <header combine="replace">
    <name>Accept</name>
    <value>application/sparql-results+xml</value>
  </header>
</submission>
```

11.8.1 The name Element

When the `name` element appears as a child of element `header`, it is used to specify the name of a header entry to be provided to the submission protocol.

Common Attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the header name, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation fails.

The header entry name may be given by the string content of the `name` element, or by the result of the `value` attribute. If both are given, the result from the `value` attribute takes precedence. If the resulting name is the empty string, then the entry is considered to be void, and it is not supplied to the submission protocol.

11.8.2 The value Element

When the `value` element appears as a child of element `header`, it is used to specify the value component of a header entry to be supplied to the submission protocol to be added to the preamble of a submission. The `value` element may be used more than once in a given element `header`, in which case each value produces a new header entry.

Common Attributes: None

Special Attributes:

value

Author-optional attribute containing an XPath expression to evaluate using the in-scope evaluation context. To obtain the header entry value, the result of the expression is processed as if by call to the XPath `string` function. An empty string is used if the XPath evaluation

fails.

The header entry value may be given by the string content of the `value` element, or by the result of the `value` attribute. If both are given, the result from the `value` attribute takes precedence.

Note:

The XPath `string` function combines multiple nodes by concatenating them into a string separated with spaces. As a result, a header value specified by a `value` element nodeset containing multiple nodes may not be properly serialized in a submission protocol preamble. To assure proper delivery of individual header items to the submission protocol, restrict use of XPath expressions producing nodesets for element `header` with attribute `nodeset`, where each node will produce its own separate header entry, and use expressions resulting in only a single node in element `value`.

11.9 Submission Options

The XForms Model specifies a `submission` element containing the following attributes and child elements that affect serialization and submission. This section summarizes the behaviors for the allowable values of these attributes and child elements, and presents subsections that define the behavior for submission and serialization.

- the [submission resource](#)
- the [submission method](#)
- the `header` elements

For the submission protocol obtained from the URI scheme in the [submission resource](#), XForms normatively defines a binding to HTTP/1.1 [\[RFC 2616\]](#).

Note:

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

The [submission method](#) determines the default data serialization format, and both the [submission method](#) and the URI scheme in the [submission resource](#) determine the submission protocol operation, according to the following table:

URI scheme	Submission Method	Default Serialization	Submission Protocol Operation
http https mailto	"post"	application/xml	HTTP POST or equivalent
http https file	"get"	application/x-www-form-urlencoded	HTTP GET or equivalent
http https file	"delete"	application/x-www-form-urlencoded	HTTP DELETE or equivalent
http https file	"put"	application/xml	HTTP PUT or equivalent
http https mailto	"multipart-post"	multipart/related	HTTP POST or equivalent
http https mailto	"form-data-post"	multipart/form-data	HTTP POST or equivalent
http https mailto	"urlencoded-post"	application/x-www-form-urlencoded	HTTP POST or equivalent
(any)	Any other NCName	application/xml	As given by the Submission Method
(any)	QNameButNotNCName	implementation-defined	implementation-defined

Note:

Foreign-namespaced attribute values are allowed in the [Submission Method](#), but no behavior is defined by XForms.

11.9.1 The get Submission Method

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

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

The URI is constructed as follows:

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

No message body is sent with the request.

Examples:

Simple search submission
<pre><submission resource="http://example.com/search" method="get"/></pre>
After doing relevance and validity checking on the data, the leaf nodes of the default instance are submitted asynchronously, encoded as a URL (application/x-www-form-urlencoded), to http://example.com/search. The result replaces the whole page.
Reading from a local file
<pre><submission resource="file:data.xml" method="get" serialization="none" replace="instance" instance="data" /></pre>

Replaces the instance 'data' with the content of the file data.xml. Serialization, and its associated validity and relevance processing, is not needed. See the corresponding example for saving a file in Section [11.9.3 The put Submission Method](#). The user agent may restrict file access to a user-specific and domain-specific security zone in local storage.

11.9.2 The post, multipart-post, form-data-post, and urlencoded-post Submission Methods

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

Examples:

Posting instance data

```
<submission resource="https://example.com/jsp/orders" method="post" ref="/purchaseOrder" />
```

Submits the XML for a purchase order to a secure server order processing system.

Simple posted login

```
<submission resource="http://example.com/login" method="urlencoded-post"/>
```

After doing relevance pruning and validity checking on the login data, the leaf nodes of the default instance are submitted asynchronously in the posted data, encoded based on the `application/x-www-form-urlencoded` serialization, to `http://example.com/login`. The result replaces the whole page.

11.9.3 The put Submission Method

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

Example:

Saving to a local file

```
<submission resource="file:data.xml" ref="instance('data')" method="put" validate="false" relevant="false" replace="none" />
```

Saves the instance 'data' to the file data.xml without validation checking and relevance pruning. See the corresponding example for reading from a local file in Section [11.9.1 The get Submission Method](#). The user agent may restrict file access to a user-specific and domain-specific security zone in local storage.

11.9.4 The delete Submission Method

This submit method represents HTTP DELETE or the equivalent concept (such as deleting a local file). The serialized form data is delivered in the same manner as the [get](#) submission method (see [11.9.1 The get Submission Method](#)).

11.9.5 Serialization as application/xml

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

The steps for serialization are as follows:

1. An XML document is produced following the rules of the XML output method defined in [XSLT 1.0](#) section 16 and 16.1, using the values supplied as attributes of the `submission` element.
 - a. Handling of namespace nodes: The default behavior is that every namespace node is serialized according to the rules of the XML output method, so that at least one namespace declaration appears in the serialized XML for each in-scope namespace. Additional inherited namespaces are declared on the root element of the serialized XML. If, however, attribute `includenamespacesprefixes` on element `submission` is present, then all namespace declarations not visibly utilized in the instance data (as defined in [\[Exc-C14N\]](#)) and the default namespace if it is empty are excluded from the root element serialization, unless the corresponding namespace prefix is listed in the `includenamespacesprefixes` attribute. The special value `#default` represents the default namespace.
 - b. Mediatype: By default, the mediatype of the serialized XML instance is `application/xml`, but can be changed to a compatible type using element `submission` attribute `mediatype`. Authors should ensure that the type specified is compatible with `application/xml`.

11.9.6 Serialization as multipart/related

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

In this format, XML instance data is serialized as one part of the [\[RFC 2387\]](#) `multipart/related` message, using the rules as described in [11.9.5 Serialization as application/xml](#). Binary content from `xsd:anyURI` instance nodes populated by the `upload` (see [8.1.6 The upload Element](#)) control is serialized in separate parts of the [\[RFC 2387\]](#) `multipart/related` message.

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

- `multipart/related` message header requirements:

- Must contain a `type` parameter of the mediatype of the serialized XML instance.
- Must contain a `start` parameter referring to the Content-ID first body part (root).
- First body part (root) requirements:
 - Must have `Content-Type` parameter of the type specified by the `submission mediatype attribute`.
 - Content is serialized by the rules at [11.9.5 Serialization as application/xml](#).
- Subsequent part requirements:
 - One part for each node with a datatype of `xsd:anyURI` populated by `upload` with:
 - A `Content-Type` header that represents the type of the attachment if known, otherwise `application/octet-stream`.
 - A `Content-Transfer-Encoding` header.
 - A `Content-ID` header whose value matches the URI in the associated instance data node.
 - The binary content associated with the URI, serialized according to the `Content-Transfer-Encoding` heading.

multipart/related

```
<submission method="multipart-post" resource="http://example.com/photo" />
```

Submits the instance data in `multipart/related`, along with the selected file as an attachment.

```
POST /photo HTTP/1.0
Host: example.com
Content-Type: multipart/related; boundary=f93dcbA3; type=application/xml; start="<980119.X53GGT@example.com>"
Content-Length: xxx

--f93dcbA3
Content-Type: application/xml; charset=UTF-8
Content-ID: <980119.X53GGT@example.com>

<?xml version="1.0"?>
<uploadDocument>
  <title>My Proposal</title>
  <author>E. X. Ample</author>
  <summary>A proposal for a new project.</summary>
  <notes image="cid:980119.X17AXM@example.com">(see handwritten region)</notes>
  <keywords>project proposal funding</keywords>
  <readonly>false</readonly>
  <filename>image.png</filename>
  <content>cid:980119.X25MNC@example.com</content>
</uploadDocument>
--f93dcbA3
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <980119.X25MNC@example.com>

...Binary data here...
--f93dcbA3
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <980119.X17AXM@example.com>

...Binary data here...
--f93dcbA3--
```

11.9.7 Serialization as multipart/form-data

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

Note:

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

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

- Each element node is visited in document order, except non-relevant elements are skipped if the `relevant` setting of the `submission` is `true`.
- Each visited element that has no child element nodes (i.e., each leaf element node) is selected for inclusion, including those that have no value (no text node).
- Element nodes selected for inclusion are encoded as `Content-Disposition: form-data` MIME parts as defined in [RFC 2388](#), with the `name` parameter being the element local name.
- Element nodes of any datatype populated by `upload` also have a `Content-Disposition filename` parameter, if the filename is available.
- Element nodes of any datatype populated by `upload` are serialized as the specified binary content. In the case of `xsd:anyURI` and derived

types, the serialization content is obtained from the URI. For `xsd:base64Binary`, `xsd:hexBinary`, and derived types, the serialization content is obtained by decoding the element string value.

- Element nodes of any datatype not populated by `upload` are serialized as the string value of the element (the concatenation of all text node children, or empty string if the element has no text node children).
- The `Content-Type` must be `text/plain` except for `xsd:anyURI`, `xsd:base64Binary`, `xsd:hexBinary`, and derived types, in which case the header represents the media type of the attachment if known, otherwise `application/octet-stream`. If a character set is applicable, the `Content-Type` may have a `charset` parameter.

Example:

multipart/form-data <pre><submission method="form-data-post" resource="http://example.com/photo" /></pre>
Submits the instance data in <code>multipart/form-data</code> , along with the selected file as a part. <pre>POST /photo HTTP/1.0 Host: example.com Content-Type: multipart/form-data; boundary=AaB03x Content-Length: xxx --AaB03x Content-Disposition: form-data; name="document"; filename="b.txt" Content-Type: text/plain; charset=iso-8859-1 This is a file. It has two lines. --AaB03x Content-Disposition: form-data; name="title" A File --AaB03x Content-Disposition: form-data; name="summary" This is my file file test --AaB03x--</pre>

11.9.8 Serialization as `application/x-www-form-urlencoded`

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

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

The steps for serialization are as follows:

1. Each element node is visited in document order, except non-relevant elements are skipped if the `relevant` setting of the `submission` is `true`. Each visited element that has no child element nodes (i.e., each leaf element node) is selected for inclusion, including those that have no value (no text node). Note that attribute information is not preserved.
2. Element nodes selected for inclusion are encoded as `EltName=value`, where `=` is a literal character, `EltName` represents the element local name, and `value` represents the string value of the element (the concatenation of all text node children, or empty string if the element has no text node children). The separator character `{sep}` from the `separator` attribute on `submission` is used between pairs of encoded name/value pairs, e.g. `EltName1=value1{sep}EltName2=value2{sep}EltName3=value3`. Note that contextual path information is not preserved, nor are namespaces or namespace prefixes. As a result, different elements might serialize to the same name.
 - The encoding of `EltName` and `value` are as follows: space characters are replaced by `+`, and then non-ASCII and reserved characters (as defined by [RFC 2396](#) as amended by subsequent documents in the IETF track) are escaped by replacing the character with one or more octets of the UTF-8 representation of the character, with each octet in turn replaced by `%HH`, where `HH` represents the uppercase hexadecimal notation for the octet value and `%` is a literal character. Line breaks are represented as "CR LF" pairs (i.e., `%0D%0A`).
3. All such encodings are concatenated, maintaining document order.

Example:

application/x-www-form-urlencoded <pre>GivenName=Ren%C3%A9</pre>
This format consists of simple name-value pairs. <pre><PersonName title="Mr"> <GivenName>René</GivenName> </PersonName></pre>
Here is the instance data for the above example. Note that very little of the data is preserved. Authors desiring greater data integrity should select a different serialization format.

11.10 Replacing Data with the Submission Response

The `submission` element allows an author-optional attribute named `targetref`. The attribute value is interpreted as a [binding expression](#) to which the first node rule is applied to obtain a **replacement target node** for the submission response. This attribute is ignored unless the value of the `replace` attribute is "instance" or "text".

For backwards compatibility with documents created for earlier versions of the specification, the processor of the `submission` element may allow the author-optional attribute named `target` to be used. The `target` attribute provides exactly the same behaviors as the `targetref` attribute except that the `target` attribute is ignored if the `submission` element also bears a `targetref` attribute.

The default replacement target node is the document element node of the instance identified by the `instance` attribute, which is equal to the default instance of the model if not specified. The evaluation context for this attribute is the in-scope evaluation context for the `submission` element, except the context node is modified to be the document element of the instance identified by the `instance` attribute if it is specified.

This attribute is evaluated only once a successful submission response has been received and if the `replace` attribute value is "instance" or "text". The first node rule is applied to the result.

The processing of the `targetref` attribute (and its default) is considered to have failed if the result is any of the following:

- an empty nodeset
- a readonly node, if `replace="text"`
- a non-element, if `replace="instance"`
- a node whose parent is readonly, if `replace="instance"`

If the processing of the `targetref` attribute fails, then submission processing ends after dispatching the event `xforms-submit-error` with an error-type of `target-error`.

If the `replace` attribute contains the value "text" and the submission response conforms to an XML mediatype (as defined by the content type specifiers in [\[RFC 3023\]](#)) or a text media type (as defined by a content type specifier of `text/*`), then the response data is encoded as text and replaces the content of the replacement target node.

If the `replace` attribute contains the value "instance" and the submission response conforms to an XML mediatype (as defined by the content type specifiers in [\[RFC 3023\]](#)) and the XML parse of the submission response succeeds, then the XML obtained from the submission response is used to replace the target node. The XML in the response may have comment and processing instruction nodes before and after the document element. These nodes are discarded if the replacement target node is not the document element of an instance. Otherwise, those processing instructions and comments replace any processing instructions and comments that previously appeared outside of the document element of the instance being replaced.

In the case of text replacement of the content of the replacement target node, the replacement is performed by the XForms Action `setvalue` ([10.2 The setvalue Element](#)). In the case of instance node replacement, the replacement is performed by an XForms action that performs some combination of node insertion and deletion operations that are performed by the `insert` action ([10.3 The insert Element](#)) and the `delete` action ([10.4 The delete Element](#)). If the `submission` has a mode of "asynchronous", then the text replacement action or the instance node replacement action is an outermost action handler, so the [deferred update behavior](#) occurs at the end of the action. If the mode is "synchronous", then the text replacement action or the instance node replacement action is not outermost since occurs during the default processing of `xforms-submit`, so the appropriate [deferred update flags](#) are set based on whether the action was a `setvalue` or whether it performed a series of `insert` and `delete` actions.

Note:

In an asynchronous submission, the deferred update behavior ensures that the user interface is up to date with the latest calculated values before the `xforms-submit-done` event is dispatched. In a synchronous submission, the calculated values dependent on replaced text or data nodes can be made available to actions in the `xforms-submit-done` handler by first invoking the `recalculate` action. A sequence of synchronous submissions performed with successive `send` actions can avoid refreshing the user interface until after the completion of the last `send` action.

Examples:

Replacing a subtree of instance data

```
<submission resource="http://example.com/jsp/prefill" method="post" ref="name" replace="instance" targetref="address"/>
```

This submission would be invoked after the user enters a value for `name`. Based on the name given, a simple server-side database lookup is performed to get a last known address. The `address` element is replaced with the result, prefiling part of the form for the user.

Replacing text in an instance

```
<submission resource="http://example.com/postalCodeSearch" method="get" ref="address" replace="text" targetref="address/postalCode"/>
```

The address information is past to a postal code search service that returns a textual result, which is placed into the `postalCode` element.

Submission and Read-Only Content

```
<model xmlns:my="http://example.org">
  <instance>
    <my:data>
      <my:name>
        <my:first-name>John</my:first-name>
        <my:last-name>Doe</my:last-name>
      </my:name>
      <my:address>
        <my:street>123 Main St.</my:street>
        <my:city>Smallville</my:city>
      </my:address>
    </my:data>
  </instance>

  <bind nodeset="/my:data/my:name/" readonly="true()"/>
  <bind nodeset="/my:data/my:address/my:street" readonly="true()"/>

  <submission id="S1" targetref="my:name" replace="instance" method="post" resource="...">
  <submission id="S2" targetref="my:name/my:first-name" replace="instance" method="post" resource="...">
```

```
<submission id="S3" targetref="my:name/my:first-name" replace="text" method="post" resource="..."/>
<submission id="S4" targetref="my:address/my:street" replace="text" method="post" resource="..."/>
</model>
```

Submission S1 succeeds because a readonly node (`my:name`) can be replaced if its parent is not readonly. Submission S2 fails because a node (`my:first-name`) cannot be replaced if its parent is readonly. Submission S3 fails because the content of a readonly node cannot be replaced, even if it is readonly due to inheritance. Submission S4 fails because the content of a readonly node cannot be replaced, even if the node's parent is not readonly.

11.11 Integration with SOAP

This section describes the integration of XForms submission with [\[SOAP 1.1\]](#) and [\[SOAP 1.2\]](#)

11.11.1 Representation of SOAP Envelope

The single-node binding of the `submission` element refers to the XML data to be submitted. In the case of a SOAP submission, the instance data includes the SOAP envelope and related SOAP tags.

Note:

The form author may choose to store the data payload in one instance and copy the data to the submission instance containing the SOAP envelope as part of an `xforms-submit` event handler. The form author is responsible for declaring the appropriate model item properties on both instances (e.g. the relevant declarations).

11.11.2 Indicating a SOAP submission

For a SOAP submission, the `mediatype` attribute of the `submission` [must](#) be set to the MIME type of `application/soap+xml`. The form author may append `charset` and `action` MIME parameters.

Note:

The `action` MIME parameter has no effect unless the submission `method` is "post" because the GET method implies no SOAP processing by the receiving SOAP node.

Note:

SOAP 1.1 does not support the HTTP GET operation.

11.11.3 SOAP HTTP Binding

The `method` attribute of the `submission` [must](#) be set to `get` or `post` in order to access the SOAP HTTP binding.

If `method="get"`, then the SOAP response message exchange pattern is used. The HTTP headers [must](#) contain the `Accept` parameter with a value conforming to the following properties:

- [must](#) begin with `application/soap+xml`
- If the submission `mediatype` contains a `charset` MIME parameter, then it is appended to the `application/soap+xml` MIME type. Otherwise, a `charset` MIME parameter with same value as the `encoding` attribute (or its default) is appended to the `application/soap+xml` MIME type.
- No other MIME parameters from the `mediatype` are copied to the `application/soap+xml` MIME type
- The `q` MIME parameter [must not](#) be specified in the `application/soap+xml` MIME type so that the default quality of 1 is used.

If `method="post"`, then the SOAP request-response message exchange pattern is used. For SOAP 1.2, the current submission behavior of using the `mediatype` attribute value as the value of the `Content-type` parameter in the HTTP headers is sufficient. If the instance data being submitted has as its root element node a SOAP envelope in the SOAP 1.1 namespace (`http://schemas.xmlsoap.org/soap/envelope/`), then:

- the `Content-type` HTTP header is changed to `text/xml`
- the `charset` MIME parameter is appended. The `charset` parameter value from the `mediatype` attribute is used if it is specified. Otherwise, the value of the `encoding` attribute (or its default) is used.
- if the `action` MIME parameter appears in the `mediatype` then a `SOAPAction` HTTP header is added and given a value equal to the content of the `action` MIME parameter

Note:

XForms 1.1 does not support the SOAP email binding, so `method="post"` with a `mailto:` scheme results in an `xforms-submit-error` event before any submit processing message is dispatched.

Note:

XForms 1.1 does not support the SOAP 1.1 binding to the HTTP Extension Framework.

Example:

Consuming a SOAP 1.1 Request-Response Web Service

```
<xforms:model xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://www.webservice.net">
  <xforms:instance id="data">
```

```

<data xmlns="">
  <city>Victoria</city>
  <country>Canada</country>
  <weather>Mostly sunny and cool. High 12C. Low 3C.</weather>
</data>
</xforms:instance>

<xforms:instance id="GetWeatherSoapIn">
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <GetWeather xmlns="http://www.webservice.net">
        <CityName>Victoria</CityName>
        <CountryName>Canada</CountryName>
      </GetWeather>
    </soap:Body>
  </soap:Envelope>
</xforms:instance>
<xforms:instance id="GetWeatherSoapOut">
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <GetWeatherResponse xmlns="http://www.webservice.net">
        <GetWeatherResult>Mostly sunny and cool. High 12C. Low 3C.</GetWeatherResult>
      </GetWeatherResponse>
    </soap:Body>
  </soap:Envelope>
</xforms:instance>

<xforms:submission id="GetWeather" resource="http://www.webservice.net/getweather.aspx" method="post"
  ref="instance('GetWeatherSoapIn')" mediatype="application/soap+xml; action=http://www.webservice.net/GetWeather"
  replace="instance" instance="GetWeatherSoapOut">
  <xforms:action ev:event="xforms-submit">
    <xforms:setvalue ref="instance('GetWeatherSoapOut')/soap:Body/tns:GetWeather/tns:CityName"
      value="instance('data')/city"/>
    <xforms:setvalue ref="instance('GetWeatherSoapOut')/soap:Body/tns:GetWeather/tns:CountryName"
      value="instance('data')/country"/>
  </xforms:action>
  <xforms:action ev:event="xforms-submit-done">
    <xforms:setvalue ref="instance('data')/weather"
      value="instance('GetWeatherSoapOut')/soap:Body/tns:GetWeatherResponse/tns:GetWeatherResult"/>
  </xforms:action>
</xforms:submission>
</xforms:model>

<xforms:input ref="city">
  <xforms:label>City </xforms:label>
  <xforms:send ev:event="xforms-value-changed" submission="GetWeather"/>
</xforms:input>
<xforms:input ref="country">
  <xforms:label>Country </xforms:label>
  <xforms:send ev:event="xforms-value-changed" submission="GetWeather"/>
</xforms:input>
<xforms:output ref="weather">
  <xforms:label>The weather forecast is </xforms:label>
</xforms:output>

```

This form accepts input of a city name and country name from the user. When the user changes either value, the 'GetWeather' web service is initiated. On `xforms-submit`, the user input is copied into the request envelope. When the web service submission result is received, the `xforms-submit-done` handler copies the weather forecast from the response envelope to the data instance.

The submission and the request and response instances correspond to the web service definition below:

```

<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.webservice.net" targetNamespace="http://www.webservice.net">
  <wsdl:types>
    <xs:schema elementFormDefault="qualified" targetNamespace="http://www.webservice.net">
      <xs:element name="GetWeather">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="CityName" type="xs:string" />
            <xs:element minOccurs="0" maxOccurs="1" name="CountryName" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="GetWeatherResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      ...
    </wsdl:types>

    <wsdl:message name="GetWeatherSoapIn">
      <wsdl:part name="parameters" element="tns:GetWeather" />
    </wsdl:message>
    <wsdl:message name="GetWeatherSoapOut">
      <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
    </wsdl:message>
    ...
    <wsdl:portType name="GetWeatherSoap">
      <wsdl:operation name="GetWeather">
        <wsdl:input message="tns:GetWeatherSoapIn" />
        <wsdl:output message="tns:GetWeatherSoapOut" />
      </wsdl:operation>
    </wsdl:portType>
  </wsdl:definitions>

```



```

...
</wsdl:portType>
...
<wsdl:binding name="GetWeatherSoap" type="tns:GetWeatherSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetWeather">
    <soap:operation soapAction="http://www.webservice.net/GetWeather" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  ...
</wsdl:binding>

<wsdl:service name="GetWeatherService">
  <wsdl:port name="GetWeatherSoap" binding="tns:GetWeatherSoap">
    <soap:address location="http://www.webservice.net/getweather.asmx" />
  </wsdl:port>
  ...
</wsdl:service>
</wsdl:definitions>

```

11.11.4 Handling the SOAP Response

The XForms processor [must](#) handle client authorization and redirection.

SOAP faults (400 and 500 level errors) are handled in the same manner as underlying HTTP errors, which is to say that an `xforms-submit-error` event is dispatched.

On successful completion, the results are consumed according to the XForms submission process, culminating in an `xforms-submit-done` event. The form author may capture this event and copy data from the target instance that receives the returned SOAP envelope to other instances that are designed to carry only data.

12 Conformance

12.1 Conforming XForms Documents

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

- [\[XML 1.0\]](#)
- [\[XML Names\]](#)
- [\[XPath 1.0\]](#)
- [\[XML Events\]](#)
- [\[XML Schema part 2\]](#)

XForms elements are typically inserted into a containing document in multiple places. The root element for each individual fragment of XForms [must](#) be `model`, a [core form control](#), `group`, `repeat`, or `switch`. Individual XForms fragments [must](#) be schema-valid according to the Schema for XForms ([F Schema for XForms](#)), except that the host language [may](#) add elements of its own namespace to the content models of the following elements: `group`, `repeat`, `case`, `label`, `help`, `hint`, `alert` and `message`.

A [host language may](#) introduce additional conformance requirements.

12.2 Conforming XForms Generators

XForms generators [should](#) generate conforming XForms documents.

12.3 Base Technologies for XForms Processors

The following are base technologies for XForms. An [XForms Processor](#) therefore [must](#) conform to the following specifications, except as qualified below:

- [\[XML 1.0\]](#)
- [\[XML Names\]](#)
- [\[XPath 1.0\]](#)
- [\[XML Events\]](#) (not all aspects are [required](#))
- [\[XML Schema part 2\]](#)

12.4 Conformance Levels

XForms is a generalized XML processing language. Therefore, the XForms specification is intended for implementation on hardware platforms of all sizes, from tiny hand-held devices to high-powered servers. This section describes two main levels of conformance that are useful in varied scenarios.

12.4.1 XForms Model

This conformance level is distinguished by the processor's `property()` function returning a string beginning with "model" for the `conformance-level` property.

An XForms `model` can contain or reference XML schemas, XForms `instance` elements, XForms `bind` elements, XForms `submission` elements, and XForms actions. XForms `submission` elements can also contain XForms actions.

An **XForms Model Processor** is a reduced functionality [XForms Processor](#) with required, recommended and optional features described in this section.

An [XForms Model Processor](#) **must** support all attributes of the `model` element. The processor **must** support the valid attributes and content of the `instance` element, and it **should** support the notification events `xforms-insert` and `xforms-delete`. The processor **must** support all attributes, content and other aspects of `bind` elements, except `p3ptype` support is **optional**. The processor **must** fully support [XPath 1.0](#), including all XForms extension functions (though the `index()` function **may** return 1 if the processor has no information about the identified `repeat` element). The processor **must** make available the automatic schema datatype information defined by XForms. The processor **must** be able to parse inline and external declared XML schema and consume their schema datatype information [\[XML Schema part 2\]](#), and the processor **should** consume all schema information available where appropriate in the XForms processing model. An [XForms Model Processor](#) **may** (and hence may not) support user interface creation and refresh behaviors described for `model` processing. An [XForms Model Processor](#) **must** support action handlers for the events `xforms-model-construct-done`, `xforms-ready`, `xforms-link-exception` and `xforms-version-exception`. The processor **may** support action handlers for the `xforms-refresh` event, and it **should** support all other events targetted at `model` elements. The support for attributes and elements of [\[XML Events\]](#) is described below in the description of the support for XForms Actions.

An [XForms Model Processor](#) **should** support the [XForms Submission module](#). If it does, then all attributes, child elements, behaviors and events **must** be supported except as follows. The support for attributes and elements of [\[XML Events\]](#) on action handlers for submission events is described below in the description of the support for XForms Actions. All `NCName` submission methods **should** be supported, and the following methods **must** be supported: `get`, `post`, `put`, `delete`, and `urlencoded-post`. The `mode` attribute **should** be supported; at least one of asynchronous or synchronous submission **must** be supported. The `http` submission scheme **must** be supported. The `https` and `file` schemes **should** be supported. Other schemes **may** be supported such as `mailto` or `ftp`. XForms-defined submission headers **may** be combined with those from the user agent in the manner specified by the `combine` attribute.

An [XForms Model Processor](#) **must** support the following actions: `action`, `insert`, `delete`, `setvalue`, `reset`, `rebuild`, `recalculate`, `revalidate`, and `dispatch`. The processor **should** support `send` and `load`. The processor **may** support `refresh`, `setindex`, `setfocus`, `toggle`, and `message`. For every supported action, the processor **must** support all local attributes defined for the action, including conditional and iteration attributes. An [XForms Model Processor](#) **must** support the attribute `ev:event` so that XForms action handlers can appear as children of either the target elements of the events they handle or ancestors of those elements. The processor **should** support the attribute `ev:target` so action handlers can identify the events targetted at a particular descendant of the action handler's parent element (e.g. the `xforms-insert` or `xforms-delete` event on a particular `instance` of the `model`). The processor **may** support all other features of [\[XML Events\]](#).

12.4.2 XForms Full

This conformance level is distinguished by the processor's `property()` function returning a string beginning with "full" for the `conformance-level` property.

An **XForms Full Processor** is an [XForms Processor](#) consisting of a conforming [XForms Model Processor](#) along with the following additional required, recommended and optional features:

- The processor **must** support user interface creation and refresh behaviors described for `model` processing.
- The processor **must** support all [core form controls](#), including all of their attributes and child elements.
- The processor **must** support the actions `refresh`, `setfocus`, and `message`.
- The processor **should** support the [XForms Group Module](#). If the processor does not support this module and a `group` element is encountered during user interface initialization, then the processor **must** terminate processing after dispatching [xforms-binding-exception](#).
- The processor **should** support the `toggle` action and the [XForms Switch Module](#). If the processor does not support this module and a `switch` element is encountered during user interface initialization, then the processor **must** terminate processing after dispatching [xforms-binding-exception](#).
- The processor **should** support the `setindex` action and the [XForms Repeat Module](#), except that support of the `repeat-*` attributes is **optional**. If the processor does not support this module and a `repeat` element is encountered during user interface initialization, then the processor **must** terminate processing after dispatching [xforms-binding-exception](#).
- The processor **should** support all interaction and notification events targetted at basic form controls, `group`, `switch`, `repeat`, and their descendant elements (e.g. `case` and `item`).
- The processor **may** support the Extension module.

13 Glossary Of Terms

Binding

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

Binding expression

[Definition: An [XPath 1.0](#) expression used in a binding.]

Compound Document

A [\[CDRF 1.0\]](#) Compound Document is a document that combines multiple document formats either by reference, by inclusion or both.

Computed expression

[Definition: An [XPath 1.0](#) expression used by model item properties such as `relevant` and `calculate` to include dynamic functionality in XForms.]

Containing document

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

Datatype

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

Facet

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

First node rule

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

Form control

[Definition: An XForms user interface control that serves as a point of user interaction (a [core form control](#)) or as a container for other form controls (a [container form control](#)).]

Host language

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

Instance data

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

Instance data node

[Definition: An [XPath 1.0](#) node from the instance data.]

Lax schema processing

[Definition: From XML Schema [XML Schema part 1](#): For an element or attributes Schema validity to be assessed, then the applicable schema must provide a definition of the item. If not, Schema validation makes no contribution to the validity test for the item.]

Lexical space

[Definition: From XML Schema [XML Schema part 2](#): A lexical space is the set of valid literals for a datatype.] The XML serialization that may occur during submission expresses the instance data using lexical space literals.

Model Binding expression

[Definition: An [XPath 1.0](#) expression used in the `nodeset` attribute of a `bind` element in an XForms model. Often, a `bind` also declares [computed expressions](#) for model item properties of the nodes.]

Model item

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

Model item property

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

Non-relevant Form Control

[Definition: A form control satisfying at least one of the [form control non-relevance conditions](#).]

QNameButNotNCName

[Definition: A [QName](#) that is not an [NCName](#). In 2006, the W3C named this a [PrefixedName](#).]

Relevant Form Control

[Definition: A form control satisfying none of the [form control non-relevance conditions](#).]

Schema constraint

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

Strict schema processing

[Definition: From XML Schema [XML Schema part 1](#): If the applicable schema does not provide a definition for an element or attribute, then Schema validation marks the item as invalid.]

UI Binding Expression

[Definition: An [XPath 1.0](#) expression used in binding a [form control](#) to the instance.]

Valid node

[Definition: An instance data node is valid if and only if the constraint model item property is true, the value is non-empty if the required model item property is true, and the node satisfies all applicable XML Schema definitions (including those associated by the type model item property, by `xsi:type` or by an external or inline schema).]

Value space

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

versionList

[Definition: A list ([\[XML Schema part 2\]](#)) with an atomic datatype ([\[XML Schema part 2\]](#)) of [versionNumber](#).]

versionNumber

[Definition: A string consisting of a non-zero digit (1 to 9) followed by zero or more digits (0 to 9), then a period character (.), and then one or more digits (0-9). A version number is derived from string by restriction based on the following pattern value (excluding the quote marks): "[1-9]\d*\.\d+".]

XForms Model

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

XForms Processor

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

A References

A.1 Normative References

Exc-C14N

[Exclusive XML Canonicalization Version 1.0](#), J. Boyer, D. Eastlake 3rd, J. Reagle, 2002. W3C Recommendation available at <http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>.

HMAC

[RFC 2104 - HMAC: Keyed-Hashing for Message Authentication](#), H. Krawczyk, M. Bellare, R. Canetti, 1997. Available at <http://www.ietf.org/rfc/rfc2104.txt>

Luhn Patent

Computer for Verifying Numbers, H. P. Luhn, U.S. Patent 2,950,048, 1960.

MD5

[RFC 1321: The MD5 Message-Digest Algorithm](#), R. Rivest, 1992. Available at <http://www.ietf.org/rfc/rfc1321.txt>

RFC 2119

[RFC 2119: Key words for use in RFCs to Indicate Requirement Levels](#), S. Bradner, 1997. Available at <http://www.ietf.org/rfc/rfc2119.txt>.

RFC 2387

[RFC 2387: The MIME Multipart/Related Content-type](#), E. Levinson, 1998. Available at: <http://www.ietf.org/rfc/rfc2387.txt>.

RFC 2388

[RFC 2388: Returning Values from Forms: multipart/form-data](#), L. Masinter, 1998. Available at: <http://www.ietf.org/rfc/rfc2388.txt>.

RFC 2396

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

RFC 2616

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

RFC 2822

[RFC 2822: Internet Message Format](#), P. Resnick, 2001. Available at: <http://www.ietf.org/rfc/rfc2822.txt>.

RFC 3023

[RFC 3023: XML Media Types](#), M. Murata, S. St. Laurent, D. Kohn, 2001. Available at: <http://www.ietf.org/rfc/rfc3023.txt>.

SHA2

[SECURE HASH STANDARD. FIPS PUB 180-2](#), August 2002. Available at <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>

SOAP 1.1

[Simple Object Access Protocol \(SOAP\) 1.1](#), D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, D. Winer, 2000. Available at: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

SOAP 1.2

[SOAP Version 1.2 Part 1: Messaging Framework](#), M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon, 2007. Available at: <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.

Unicode Collation Algorithm

[Unicode Technical Standard #10. Unicode Collation Algorithm](#), Available at: <http://www.unicode.org/unicode/reports/tr10/>.

XHTML Modularization

[XHTML Modularization 1.1](#), D. Austin, S. Peruvemba, S. McCarron, M. Ishikawa, M. Birbeck, 2008. W3C Recommendation available at <http://www.w3.org/TR/2008/REC-xhtml-modularization-20081008>.

XML Base

[XML Base \(Second Edition\)](#), J. Marsh, R. Tobin, 2009. W3C Recommendation available at: <http://www.w3.org/TR/2009/REC-xmlbase-20090128/>.

XML Events

[XML Events - An events syntax for XML](#), Steven Pemberton, T. V. Raman, Shane P. McCarron, 2003. W3C Recommendation available at: <http://www.w3.org/TR/2003/REC-xml-events-20031014/>.

XHTML 1.0

[XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0](#), Steven Pemberton, et al., 2002.

W3C Recommendation available at: <http://www.w3.org/TR/2002/REC-xhtml1-20020801/>.

XML 1.0

[Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)](#), Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 2006. W3C Recommendation available at: <http://www.w3.org/TR/2006/REC-xml-20060816/>

XML Names

[Namespaces in XML \(Second Edition\)](#), Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin, 2006. W3C Recommendation available at: <http://www.w3.org/TR/2006/REC-xml-names-20060816/>.

XPath 1.0

[XML Path Language \(XPath\) Version 1.0](#), James Clark, Steve DeRose, 1999. W3C Recommendation available at: <http://www.w3.org/TR/1999/REC-xpath-19991116/>.

XML Schema part 1

[XML Schema Part 1: Structures](#), Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2004. W3C Recommendation available at: <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.

XML Schema part 2

[XML Schema Part 2: Datatypes](#), Paul V. Biron, Ashok Malhotra, 2004. W3C Recommendation available at: <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.

XSLT 1.0

[XSL Transformations \(XSLT\) Version 1.0](#), James Clark, 1999. W3C Recommendation available at: <http://www.w3.org/TR/1999/REC-xslt-19991116/>.

A.2 Informative References

Algorithms

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

AUI97

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

CDRF 1.0

[Compound Document by Reference Framework 1.0](#), Timur Mehrvarz, Lasse Pajunen, Julien Quint, and Daniel Applequist, 2007. W3C Candidate Recommendation available at: <http://www.w3.org/TR/2007/CR-CDR-20070718/>.

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/1998/REC-CSS2-19980512/>.

DDJ-ArrayDoubling

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

DOM2 Core

[Document Object Model \(DOM\) Level 2 Core Specification](#), Tom Pixley, 2000. W3C Recommendation available at: <http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/>.

DOM2 Events

[Document Object Model \(DOM\) Level 2 Events Specification](#), Tom Pixley, 2000. W3C Recommendation available at: <http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/>.

EXSLT

[EXSLT Web site](#). Available at <http://www.exslt.org>.

ODF 1.1

[Open Document Format for Office Applications \(OpenDocument\) v1.1](#), Patrick Durusau, Michael Brauer, and Lars Oppermann (editors), 2007. OASIS Standard available at: <http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1.html#OpenDocument-v1.1.html>.

P3P 1.0

[The Platform for Privacy Preferences 1.0 \(P3P1.0\) Specification](#), Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, Joseph Reagle, 2002. W3C Recommendation available at: <http://www.w3.org/TR/2002/REC-P3P-20020416/>.

RELAXNG

[RELAXNG Specification](#), James Clark, MURATO Makoto, 2001. OASIS Committee Specification available at: <http://www.relaxng.org/spec-20011203.html>.

RELAXNG Compact

[RELAXNG Compact Syntax](#), James Clark, 2002. OASIS Committee Specification available at: <http://www.relaxng.org/compact-20021121.html>.

SVG 1.1

[SVG 1.1](#), Jon Ferraiolo, FUJISAWA Jun, Dean Jackson, 2003. W3C Recommendation available at: <http://www.w3.org/TR/2003/REC-SVG11-20030114/>.

TAG Finding 7

[TAG Finding: URIs, Addressability, and the use of HTTP GET](#), Ian Jacobs, 2004. Available at: <http://www.w3.org/2001/tag/doc/whenToUseGet-20040321>

Unicode Script Names

[ISO 15924: Codes for the representation of names of scripts](#), Available at: <http://unicode.org/iso15924/iso15924-codes.html>.

UAAG 1.0

[User Agent Accessibility Guidelines 1.0](#), Ian Jacobs, Jon Gunderson, Eric Hansen, 2002. Working Draft available at <http://www.w3.org/TR/UAAG10/>.

Unicode Scripts

[Script Names](#), Mark Davis, 2001. Unicode Technical Report #24 available at <http://www.unicode.org/unicode/reports/tr24/>.

XForms 1.0

[XForms 1.0 Third Edition](#), John Boyer, 2007. W3C Recommendation available at: <http://www.w3.org/TR/2007/REC-xforms-20071029/>.

XForms Basic

[XForms Basic Profile](#), Micah Dubinko, T. V. Raman, 2003. W3C Candidate Recommendation available at: <http://www.w3.org/TR/2003/CR-xforms-basic-20031014/>.

XLink 1.0

[XML Linking Language \(XLink\) Version 1.0](#), Steve DeRose, Eve Maler, David Orchard, 2001. W3C Recommendation available at: <http://www.w3.org/TR/2001/REC-xml-20010627/>.

XPath 2.0

[XML Path Language \(XPath\) 2.0](#), Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon, 2007. W3C Recommendation available at: <http://www.w3.org/TR/2007/REC-xpath20-20070123/>.

XML Schema part 0

[XML Schema Part 0: Primer Second Edition](http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/), David C. Fallside and Priscilla Walmsley, 2004. W3C Recommendation available at: <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>.

B Patterns for Data Mutations

This appendix provides several usage patterns for the `setvalue`, `insert` and `delete` actions that perform various kinds of mutations of instance data elements and attributes.

B.1 Prepend Element Copy

Pattern: `<insert context="parent of newelement" origin="element to copy"/>`

Note:

The `context` attribute is used so that this pattern will work whether or not the parent element is empty.

Operation: Prepend a new, empty `person` element into a list of `people`

```
<xforms:insert context="people" origin="instance('prototypes')/person"/>
```

Data Before Operation

```
<xforms:instance>
  <data xmlns="">
    <people>
      <person>
        <name>Jane Doe</name>
      </person>
    </people>
  </data>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name/>
    </person>
  </prototypes>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <data xmlns="">
    <people>
      <person>
        <name/>
      </person>
      <person>
        <name>Jane Doe</name>
      </person>
    </people>
  </data>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name/>
    </person>
  </prototypes>
</xforms:instance>
```

B.2 Append Element Copy

Pattern: `<insert context="parent of newelement" nodeset="*" origin="element to copy"/>`

Note:

The `context` attribute is used so that this pattern will work whether or not the parent element is empty.

Operation: Append a new, empty `person` element into a list of `people`

```
<xforms:insert context="people" nodeset="person" origin="instance('prototypes')/person"/>
```

Data Before Operation

```
<xforms:instance>
  <data xmlns="">
    <people>
      <person>
        <name>Jane Doe</name>
      </person>
    </people>
  </data>
</xforms:instance>
```

```

        </person>
      </people>
    </data>
  </xforms:instance>

  <xforms:instance id="prototypes">
    <prototypes xmlns="">
      <person>
        <name/>
      </person>
    </prototypes>
  </xforms:instance>

```

Data After Operation

```

<xforms:instance>
  <data xmlns="">
    <people>
      <person>
        <name>Jane Doe</name>
      </person>
      <person>
        <name/>
      </person>
    </people>
  </data>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name/>
    </person>
  </prototypes>
</xforms:instance>

```

B.3 Duplicate Element

Pattern: **<insert nodeset="exact element to duplicate"/>**

Note:

The `context` attribute is not used because this pattern assumes the ability to indicate an exact element to duplicate, so `nodeset` is used. If the element does not exist, the operation will have no effect.

Operation: Duplicate the selected element and place it as a following sibling

```
<xforms:insert nodeset="paragraph[2]" />
```

Data Before Operation

```

<xforms:instance>
  <document xmlns="">
    <header>Lorem ipsum</header>
    <paragraph>Lorem ipsum verterem voluptaria ...</paragraph>
    <paragraph>Primis abhorreant delicatissimi ...</paragraph>
    <header>Lorem ipsum</header>
    <header>Lorem ipsum</header>
  </document>
</xforms:instance>

```

Data After Operation

```

<xforms:instance>
  <document xmlns="">
    <header>Lorem ipsum</header>
    <paragraph>Lorem ipsum verterem voluptaria ...</paragraph>
    <paragraph>Primis abhorreant delicatissimi ...</paragraph>
    <paragraph>Primis abhorreant delicatissimi ...</paragraph>
    <header>Lorem ipsum</header>
    <header>Lorem ipsum</header>
  </document>
</xforms:instance>

```

B.4 Set Attribute

Pattern: **<insert context="container element receiving attribute" origin="attribute to copy"/>**

Note:

The `nodeset` attribute is not used because this pattern cannot indicate an exact attribute sibling of the attribute being inserted. This is true not only because attribute order is not guaranteed but also because the attribute being inserted may already exist, in which case the existing attribute is replaced. The `context` attribute is used when it is necessary to indicate the parent of the node being inserted.

Operation: Create or replace an attribute with a copy of a given attribute

```
<xforms:insert context="item[2]" origin="../item[1]/@rating"/>
<xforms:insert context="item[3]" origin="../item[1]/@rating"/>
```

Data Before Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="23" rating="classified"/>
    <item key="42"/>
    <item key="68" rating="unknown"/>
  </items>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="23" rating="classified"/>
    <item key="42" rating="classified"/>
    <item key="68" rating="classified"/>
  </items>
</xforms:instance>
```

B.5 Remove Element

Pattern: **<delete nodeset="exact element to remove"/>**

Note:

The `context` attribute is not used because this pattern assumes the ability to indicate an exact element to remove, so `nodeset` is used. If the element does not exist, the operation will have no effect.

Operation: Remove `item` element in case it exists

```
<xforms:delete nodeset="item[2]"/>
```

Data Before Operation

```
<xforms:instance>
  <shoppingcart xmlns="">
    <item>
      <product>SKU-0815</product>
      <quantity>1</quantity>
      <unitcost>29.99</unitcost>
      <price>29.99</price>
    </item>
    <item>
      <product>SKU-4711</product>
      <quantity>3</quantity>
      <unitcost>7.49</unitcost>
      <price>22.47</price>
    </item>
  </shoppingcart>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <shoppingcart xmlns="">
    <item>
      <product>SKU-0815</product>
      <quantity>1</quantity>
      <unitcost>29.99</unitcost>
      <price>29.99</price>
    </item>
  </shoppingcart>
</xforms:instance>
```

B.6 Remove Attribute

Pattern: **<delete nodeset="exact attribute to remove"/>**

Note:

The `context` attribute is not used because this pattern assumes the ability to indicate an exact attribute to remove, so `nodeset` is used. If the attribute does not exist, the operation will have no effect.

Operation: Remove `rating` attribute in case it exists

```
<xforms:delete nodeset="item/@rating"/>
```

Data Before Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="23" rating="classified"/>
  </items>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="23"/>
  </items>
</xforms:instance>
```

B.7 Remove Nodeset

Pattern: **<delete nodeset="nodeset to remove"/>**

Note:

The `context` attribute is not used because this pattern assumes the ability to indicate a nodeset to remove, so `nodeset` is used. If the nodeset does not exist, the operation will have no effect.

Operation: Remove `track` nodeset in case it exists

```
<xforms:delete nodeset="track"/>
```

Data Before Operation

```
<xforms:instance>
  <playlist xmlns="">
    <name>Music for Airports</name>
    <track id="382"/>
    <track id="461"/>
    <track id="629"/>
  </playlist>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <playlist xmlns="">
    <name>Music for Airports</name>
  </playlist>
</xforms:instance>
```

B.8 Copy Nodeset

Pattern: **<insert context="parent of newnodeset" nodeset="*" origin="nodeset to copy"/>**

Note:

The `context` attribute is used so that this pattern will work whether or not the parent element is empty.

Operation: Append a `person` nodeset into a list of `people`

```
<xforms:insert context="people" nodeset="person" origin="instance('prototypes')/person"/>
```

Data Before Operation

```
<xforms:instance>
  <data xmlns="">
    <people/>
  </data>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name>Jane Doe</name>
    </person>
    <person>
      <name>John Doe</name>
    </person>
    <person>
      <name>Joe Sixpack</name>
    </person>
  </prototypes>
```

```
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <data xmlns="">
    <people>
      <person>
        <name>Jane Doe</name>
      </person>
      <person>
        <name>John Doe</name>
      </person>
      <person>
        <name>Joe Sixpack</name>
      </person>
    </people>
  </data>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name>Jane Doe</name>
    </person>
    <person>
      <name>John Doe</name>
    </person>
    <person>
      <name>Joe Sixpack</name>
    </person>
  </prototypes>
</xforms:instance>
```

B.9 Copy Attribute List

Pattern: `<insert context="exact element receiving attribute list" origin="attribute list to copy"/>`

Operation: Copy attribute list from one `item` to another

```
<xforms:insert context="item[2]" origin="../item[1]/@"/*"/>
```

Data Before Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="0" rating="classified"/>
    <item/>
  </items>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="0" rating="classified"/>
    <item key="0" rating="classified"/>
  </items>
</xforms:instance>
```

B.10 Replace Element

Pattern: `<insert nodeset="exact element to replace" origin="element to copy"/> <delete nodeset="exact element to replace"/>`

Note:

The `context` attribute is not used because this pattern assumes the ability to indicate an exact element to replace, so `nodeset` is used. If the element does not exist, both insert and delete operation will have no effect.

Operation: Replace a `person` element by copying a new one and removing the old one

```
<xforms:insert nodeset="person[1]" origin="instance('prototypes')/person"/>
<xforms:delete nodeset="person[1]"/>
```

Data Before Operation

```
<xforms:instance>
  <people xmlns="">
    <person>
      <name>John Doe</name>
    </person>
  </people>
</xforms:instance>
```

```
<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name/>
    </person>
  </prototypes>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <people xmlns="">
    <person>
      <name/>
    </person>
  </people>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <person>
      <name/>
    </person>
  </prototypes>
</xforms:instance>
```

B.11 Replace Attribute

Pattern: `<setvalue ref="exact attribute to replace" value="attribute to copy"/>`

Note:

If the attribute does not exist, the operation will have no effect.

Operation: Replace an attribute with the copy of a given attribute

```
<xforms:setvalue ref="item[2]/@key" value="../../../item[1]/@key"/>
```

Data Before Operation

```
<xforms:instance >
  <items xmlns="">
    <item key="0"/>
    <item key="4711"/>
  </items>
</xforms:instance>
```

Data After Operation

```
<xforms:instance >
  <items xmlns="">
    <item key="0"/>
    <item key="0"/>
  </items>
</xforms:instance>
```

B.12 Replace Instance with Insert

Pattern: `<insert nodeset="root node of instance to replace" origin="element to copy"/>`

Note:

The `context` attribute is not used because this pattern assumes the ability to indicate an instance root node to replace, so `nodeset` is used. Since an instance cannot be empty, `nodeset` will always be non-empty. Insert implements special handling for instance root nodes, thus a delete operation is not necessary.

Operation: Replace instance root node with an empty `shoppingcart` element

```
<xforms:insert nodeset="." origin="instance('prototypes')/shoppingcart"/>
```

Data Before Operation

```
<xforms:instance>
  <shoppingcart xmlns="">
    <item>
      <product>SKU-0815</product>
      <quantity>1</quantity>
      <unitcost>29.99</unitcost>
      <price>29.99</price>
    </item>
    <item>
```

```

        <product>SKU-4711</product>
        <quantity>3</quantity>
        <unitcost>7.49</unitcost>
        <price>22.47</price>
    </item>
</shoppingcart>
</xforms:instance>

<xforms:instance id="prototypes">
    <prototypes xmlns="">
        <shoppingcart/>
    </prototypes>
</xforms:instance>

```

Data After Operation

```

<xforms:instance>
    <shoppingcart xmlns=""/>
</xforms:instance>

<xforms:instance>
    <prototypes xmlns="">
        <shoppingcart/>
    </prototypes>
</xforms:instance>

```

B.13 Move Element

Pattern: **<insert context="newparent of element" nodeset="" origin="exact element to move"/> <delete nodeset="exact element to move"/>**

Note:

The `context` attribute is used for insert so that this pattern will work whether or not the new parent element is empty. For delete the `nodeset` attribute is used instead because this pattern assumes the ability to indicate an exact element to move. If the element to be moved does not exist, both insert and delete operation will have no effect.

Operation: Copy an existing element to a new parent and remove the original

```

<xforms:insert context="playlist[2]" nodeset="track" origin="../playlist[1]/track[2]"/>
<xforms:delete nodeset="playlist[1]/track[2]"/>

```

Data Before Operation

```

<xforms:instance>
    <library xmlns="">
        <playlist>
            <name>Music for Airports</name>
            <track id="382"/>
            <track id="461"/>
            <track id="629"/>
        </playlist>
        <playlist>
            <name>Lullabies</name>
            <track id="251"/>
            <track id="331"/>
        </playlist>
    </library>
</xforms:instance>

```

Data After Operation

```

<xforms:instance>
    <library xmlns="">
        <playlist>
            <name>Music for Airports</name>
            <track id="382"/>
            <track id="629"/>
        </playlist>
        <playlist>
            <name>Lullabies</name>
            <track id="251"/>
            <track id="331"/>
            <track id="461"/>
        </playlist>
    </library>
</xforms:instance>

```

B.14 Move Attribute

Pattern: **<insert context="exact element receiving attribute" origin="exact attribute to move"/> <delete nodeset="exact attribute to move"/>**

Note:

If the attribute to be moved does not exist, both insert and delete operation will have no effect.

Operation: Copy an existing attribute to a new element and remove the original

```
<xforms:insert context="item[2]" origin="../item[1]/@rating"/>
<xforms:delete nodeset="item[1]/@rating"/>
```

Data Before Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="23" rating="classified"/>
    <item key="42"/>
  </items>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <items xmlns="">
    <item key="23"/>
    <item key="42" rating="classified"/>
  </items>
</xforms:instance>
```

B.15 Insert Element into Non-Contiguous, Heterogeneous Nodeset

Pattern: `<insert nodeset="non-contiguous, heterogeneous nodeset" origin="element to copy" at="insert location"/>`

Note:

The `context` attribute is not used because it adheres to the first node rule and therefore would not allow to select a non-contiguous nodeset. The `nodeset` attribute is used instead to select a nodeset consisting of nodes with different names and different parents. The parent of the new node is the same as the parent of the insert location node selected by the combination of `nodeset` and `at`.

Operation: Copy an existing element into a non-contiguous, heterogeneous nodeset at a specified position

```
<xforms:insert nodeset="chapter/*" origin="instance('prototypes')/paragraph" at="7" position="before"/>
```

Data Before Operation

```
<xforms:instance>
  <document xmlns="">
    <chapter>
      <header>Lorem ipsum</header>
      <paragraph>Lorem ipsum verterem voluptaria ...</paragraph>
      <diagram>Exemplum 1</diagram>
      <diagram>Exemplum 2</diagram>
      <paragraph>Primis abhorreant delicatissimi ...</paragraph>
    </chapter>
    <chapter>
      <header>Lorem ipsum</header>
      <diagram>Exemplum 3</diagram>
    </chapter>
  </document>
</xforms:instance>

<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <chapter/>
    <header/>
    <paragraph/>
    <diagram/>
  </prototypes>
</xforms:instance>
```

Data After Operation

```
<xforms:instance>
  <document xmlns="">
    <chapter>
      <header>Lorem ipsum</header>
      <paragraph>Lorem ipsum verterem voluptaria ...</paragraph>
      <diagram>Exemplum 1</diagram>
      <diagram>Exemplum 2</diagram>
      <paragraph>Primis abhorreant delicatissimi ...</paragraph>
    </chapter>
    <chapter>
      <header>Lorem ipsum</header>
      <paragraph/>
      <diagram>Exemplum 3</diagram>
    </chapter>
  </document>
</xforms:instance>
```

```
<xforms:instance id="prototypes">
  <prototypes xmlns="">
    <chapter/>
    <paragraph/>
    <diagram/>
  </prototypes>
</xforms:instance>
```

C Recalculation Sequence Algorithm

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

Following is the default handling for a `recalculate` action. Action `recalculate` is defined in [10.10 The recalculate Element](#).

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

C.1 Details on Creating the Master Dependency Directed Graph

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

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

The `depList` for each vertex is assigned based on the referenced XML nodes of instance nodes, which are obtained by parsing the computed expression bound to the node (e.g., by `calculate`, `relevant`, `readOnly`, or `required`). Any expression violating any Binding Expression Constraint causes an exception ([4.5.2 The xforms-compute-exception Event](#)), terminating the `recalculate` process.

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

A computational expression appearing in a `calculate` attribute controls the text content (value) of one or more instance nodes. A vertex exists for each instance node to represent the expression in the context of the node. Likewise, computational expressions for model item properties such as `readOnly` and `required` are applied to one or more instance nodes, and vertices are created to represent such expressions in the context of each applicable node. The computational expression of each vertex must be examined to determine the XML nodes to which it refers. Any expression violating any Binding Expression Constraint causes an exception ([4.5.2 The xforms-compute-exception Event](#)), terminating the `recalculate` process. A computational expression refers to a vertex v if a subexpression indicates the `InstanceNode` for v and v represents the instance node text content (its value). In this version of XForms, model item properties such as `readOnly` and `required` cannot be referenced in an expression.

C.2 Details on Creating the Pertinent Dependency Subgraph

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

Sample Algorithm to Create the Pertinent Dependency Subgraph

This algorithm creates a pertinent dependency subgraph s from a list of changed instance data nodes $L_{\langle sub \rangle c \langle /sub \rangle}$. Variables such as v and w represent vertices in the master dependency directed graph. The same variables ending with s indicate vertices in the pertinent dependency subgraph s .

```
// Use depth-first search to explore master digraph subtrees rooted at
// each changed vertex. A 'visited' flag is used to stop exploration
// at the boundaries of previously explored subtrees (because subtrees
// can overlap in directed graphs).
for each vertex  $r$  in  $L_c$ 
  if  $r$  is not visited
  {
    Push the pair (NIL,  $r$ ) onto a stack
    while the stack is not empty
    {
      ( $v$ ,  $w$ ) = pop dependency pair from stack
      if  $w$  is not visited
      {
```



```

    Set the visited flag of  $w$  to true
    Create a vertex  $wS$  in  $S$  to represent  $w$ 
    Set the index of  $w$  equal to the array location of  $wS$ 
    Set the index of  $wS$  equal to the array location of  $w$ 
    Set the  $InstanceNode$  of  $wS$  equal to the  $InstanceNode$  of  $w$ 
    Set the type of  $wS$  equal to the type of  $w$ 
    For each dependency node  $x$  of  $w$ 
        Push the pair  $(w, x)$  onto the stack
    }
    else Obtain  $wS$  from index of  $w$ 
    if  $v$  is not NIL
    {
        Obtain  $vS$  from index of  $v$ 
        Add dependency node for  $wS$  to  $vS$ 
        Increment  $inDegree$  of  $wS$ 
    }
}

// Now clear the visited flags set in the loop above
for each vertex  $vS$  in  $S$ 
{
    Obtain  $v$  from index of  $vS$ 
    Assign false to the visited flag of  $v$ 
}

```

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

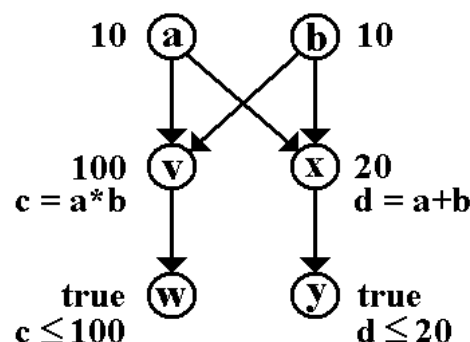
C.3 Details on Computing Individual Vertices

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

1. A vertex with $inDegree$ of 0 is selected for evaluation and removed from the pertinent dependency subgraph. In the case where more than one vertex has $inDegree$ zero, no particular ordering is specified. If the pertinent dependency subgraph contains vertices, but none have an $inDegree$ of 0, then the calculation structure of the form has a loop, and an exception ([4.5.2 The xforms-compute-exception Event](#)) must be thrown, terminating processing.
2. If the vertex corresponds to a computed item, computed expressions are evaluated as follows:
 - a. **calculate**: If the value of the model item changes, the corresponding instance data is updated and the dirty flag is set.
 - b. **relevant**, **readonly**, **required**, **constraint**: If any or all of these computed properties change, the new settings are placed into effect for associated form controls.
3. For each vertex in the `depList` of the removed vertex, decrement the $inDegree$ by 1.
4. If no vertices remain in the pertinent dependency subgraph, then the calculation has successfully completed. Otherwise, repeat this sequence from step 1.

C.4 Example of Calculation Processing

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



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

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

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

D Privacy Considerations

D.1 Using P3P with XForms

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

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

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

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

E Input Modes (Non-Normative)

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

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

E.1 `inputmode` Attribute Value Syntax

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

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

E.2 User Agent Behavior

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

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

Note:

Additional guidelines for user agent implementation are found at [\[UAAG 1.0\]](#).

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

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

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

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

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

E.3 List of Tokens

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

E.3.1 Script Tokens

Script tokens provide a general indication of the set of characters that is covered by an input mode. In most cases, script tokens correspond directly to [Unicode Scripts](#). However, this neither means that an input mode has to allow input for all the characters in the script, nor that an input mode is limited to only characters from that specific script. As an example, a "latin" keyboard doesn't cover all the characters in the Latin script, and includes punctuation which is not assigned to the Latin script.

The script tokens that are allowed are listed in [Unicode Script Names](#), "codes for the representations of scripts". The allowable values are those listed in the column "Property Value Alias" with the underscore character (`_`) removed, and excluding the two values "Common", and "Unknown". At the time of writing, these values are:

- Arabic
- Armenian
- Balinese
- Bengali
- Bopomofo
- Braille
- Buginese
- Buhid
- CanadianAboriginal
- Carian
- Cherokee
- Coptic
- Cuneiform
- Cypriot
- Cyrillic
- Deseret
- Devanagari
- Ethiopic
- Georgian
- Glagolitic
- Gothic
- Greek
- Gujarati
- Gurmukhi
- Han
- Hangul
- Hanunoo
- Hebrew
- Hiragana
- Kannada
- Katakana
- KatakanaOrHiragana
- KayahLi
- Kharoshthi
- Khmer
- Lao
- Latin

- Lepcha
- Limbu
- LinearB
- Lycian
- Lydian
- Malayalam
- Mongolian
- Myanmar
- NewTaiLue
- Nko
- Ogham
- OlChiki
- OldItalic
- OldPersian
- Oriya
- Osmanya
- PhagsPa
- Phoenician
- Rejang
- Runic
- Saurashtra
- Shavian
- Sinhala
- Sundanese
- SylotiNagri
- Syriac
- Tagalog
- Tagbanwa
- TaiLe
- Tamil
- Telugu
- Thaana
- Thai
- Tibetan
- Tifinagh
- Ugaritic
- Vai
- Yi

Seven other values are allowed:

Input Mode Token	Comments
ipa	International Phonetic Alphabet
hanja	Subset of 'han' used in writing Korean
kanji	subset of 'han' used in writing Japanese
math	mathematical symbols and related characters, representing the Unicode Script Names code "Zmth"
simplifiedHanzi	representing the Unicode Script Names code "Hans"
traditionalHanzi	representing the Unicode Script Names code "Hant"
user	special value denoting the 'native' input of the user according to the system environment

E.3.2 Modifier Tokens

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

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

E.4 Relationship to XML Schema pattern facets

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

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

E.5 Examples

This is an example of the user interface markup for a form for user input in Japanese. .

```
<xf:input ref="name" inputmode="kanji">
  <xf:label>Family name:</xf:label>
</xf:input>

<xf:input ref="nameKana" inputmode="katakana">
  <xf:label>Family name in kana:</xf:label>
</xf:input>

<xf:input ref="given" inputmode="kanji">
  <xf:label>Given name:</xf:label>
</xf:input>

<xf:input ref="givenKana" inputmode="katakana">
  <xf:label>Given name in kana:</xf:label>
</xf:input>

<xf:input ref="email" inputmode="latin lowerCase">
  <xf:label>Email:</xf:label>
</xf:input>

<xf:input ref="phone" inputmode="latin digits">
  <xf:label>Telephone:</xf:label>
</xf:input>

<xf:textarea ref="comments" inputmode="user predictOn">
  <xf:label>Comments:</xf:label>
</xf:textarea>

<xf:submit submission="sendit">
  <xf:label>Send It</xf:label>
</xf:submit>
```

F Schema for XForms (Non-Normative)

The XML Schema for XForms, which has a target namespace <http://www.w3.org/2002/xforms>, is located at <http://www.w3.org/MarkUp/Forms/2007/XForms-11-Schema.xsd>.

The RELAXNG ([RELAXNG], [RELAXNG Compact]) Schema for XForms, which includes the target namespace <http://www.w3.org/2002/xforms> as well as the version suitable for import to a host language namespace, is located at

<http://www.w3.org/MarkUp/Forms/2007/XForms-11-RELAXNG.zip>.

F.1 Schema for XML Events

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

G XForms and Styling (Non-Normative)

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

G.1 Pseudo-classes

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

Name	Defined in:	Relationship to XForms
:enabled & :disabled	[CSS3]	Selects any form control that is relevant or non-relevant (respectively).
:required & :optional	TBD	Selects any core form control bound to a node with the model item property <code>required</code> evaluating to true or false (respectively).
:valid & :invalid	TBD	Selects any core form control bound to a node that is currently valid or invalid (respectively), as defined by XForms.
:read-only & :read-write	TBD	Selects any core form control bound to a node with the model item property <code>readonly</code> evaluating to true or false (respectively).
:out-of-range & :in-range	TBD	Selects any core form control bound to a node that contains a value the form control is not or is capable of rendering, (respectively).
:value-empty & :value-non-empty	TBD	Selects any core form control bound to a node whose content is the empty string or not the empty string (respectively).

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

G.2 Pseudo-elements

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

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

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

G.3 Examples

The following examples collect together styling recommendations from this document, expressing them with a CSS notation. Throughout the examples, the following namespace declaration is assumed:

```
@namespace xf url('http://www.w3.org/2002/xforms');
```

From [8.1.1 Implementation Requirements Common to All Form Controls](#): "All form controls, including [container form controls](#), should have an inline layout by default... By default, [repeat items](#) should have a block layout." Hence, if the `display` property is not set via CSS, it should default to the following:

```
xf|input:enabled, xf|secret:enabled, xf|textarea:enabled, xf|output:enabled,
xf|upload:enabled, xf|range:enabled, xf|trigger:enabled, xf|submit:enabled,
xf|select:enabled, xf|select1:enabled { display: inline; }
xf|output:enabled { display: inline; }
xf|group:enabled, xf|switch:enabled { display: inline; }
xf|repeat:enabled { display: inline; }
::repeat-item:enabled { display: block; }
```

From [6.1.4 The relevant Property](#): "Typically, non-relevant content is not presented, or it may be styled as disabled." If the `display` property is not set via CSS, it should default to the following:

```
xf|input:disabled, xf|secret:disabled, xf|textarea:disabled,
xf|output:disabled, xf|upload:disabled, xf|range:disabled,
```

```

xf|trigger:disabled, xf|submit:disabled, xf|select:disabled,
xf|select1:disabled
{
  display: none;
}

xf|output:disabled { display: none; }
xf|group:disabled, xf|switch:disabled { display: none; }
::repeat-item:disabled { display: none; }

```

Below is an example of how a form author can override the default `display:none` styling of `:disabled`. Note that the implementation must also implement the behavior of not accepting user input in the disabled control.

```

xf|input.authorization:disabled {
  display: inline; background-color: #bbbbbb
}

```

From **8.1.1 Implementation Requirements Common to All Form Controls**: "The readonly form control should render in a way which indicates that entering or changing the value is not allowed." Below is an example of one way to indicate this information. CSS does not have a way to provide readonly behavior, so this example merely changes the background color. An implementation could use this style by default, and an author would only need to use styling to override the default.

```

xf|input.enabled:readonly::value
{
  display: inline; background-color: #888888
}

```

From **8.1.1 Implementation Requirements Common to All Form Controls**: "Except as noted, relevant form controls must distinguish rendering between being bound to a required node versus a non-required node. Exceptions are form controls that do not directly render the string value of the bound node (including trigger and the container form controls)." Below is an example styling form controls bound to required nodes with a soft yellow background. An implementation could use this style by default, and an author would only need to use styling to override the default.

```

xf|input.enabled:required, xf|secret.enabled:required,
xf|textarea.enabled:required, xf|output.enabled:required,
xf|upload.enabled:required, xf|range.enabled:required,
xf|trigger.enabled:required, xf|submit.enabled:required,
xf|select.enabled:required, xf|select1.enabled:required
{
  background-color: #FFFFD0
}

```

From **8.1.1 Implementation Requirements Common to All Form Controls**: Relevant form controls must distinguish rendering between valid and "invalid states. Control of this behavior should be made available to stylesheets." Below is an example styling input controls bound to non-empty but invalid nodes with a reddish background. An implementation could use this style by default, and an author would only need to use styling to override the default.

```

xf|input.enabled:invalid:value-non-empty {
  background-color: #ff8080;
}

```

From **8.1.1 Implementation Requirements Common to All Form Controls**: "Relevant form controls must indicate when the bound instance data contains a value or content that the form control is not capable of rendering. Control of this behavior should be made available to stylesheets". In this example, select, select1, and range controls bound nodes whose values are not presentable by the controls are styled with in red. An implementation could use this style by default, and an author would only need to use styling to override the default.

```

xf|select1.enabled:out-of-range, xf|select1.enabled:out-of-range,
xf|range.enabled:out-of-range {
  color: red;
}

```

H Complete XForms Examples (Non-Normative)

This section presents complete XForms examples. These and additional examples are maintained at <http://www.w3.org/MarkUp/Forms/2002/Examples>.

H.1 XForms in XHTML

```

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:my="http://commerce.example.com/payment" xml:lang="en">
  <head>
    <title>XForms in XHTML</title>

    <model xmlns="http://www.w3.org/2002/xforms" schema="payschema.xsd">
      <instance>
        <my:payment as="credit">
          <my:cc />
          <my:exp />
        </my:payment>
      </instance>
    </model>
  </head>
  <body>
    <my:payment />
  </body>
</html>

```



```

</instance>
<submission action="http://www.example.com/buy.rb" method="post" id="s00" />
<bind nodeset="my:cc" relevant="../@as='credit'" required="true()" />
<bind nodeset="my:exp" relevant="../@as='credit'" required="true()" />
</model>
</head>
<body>
...
<group xmlns="http://www.w3.org/2002/xforms">
  <select1 ref="@as">
    <label>Select Payment Method</label>
    <item>
      <label>Cash</label>
      <value>cash</value>
      <message level="modeless" ev:event="xforms-select">
        Please do not mail cash.</message>
    </item>
    <item>
      <label>Credit</label>
      <value>credit</value>
    </item>
  </select1>

  <input ref="my:cc">
    <label>Credit Card Number</label>
    <alert>Please specify a valid credit card number
      (use spaces or hyphens between digit groups)</alert>
  </input>

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

  <submit submission="s00">
    <label>Buy</label>
  </submit>
</group>
...
</body>
</html>

```

Schema file payschema.xsd:

```

<!-- payschema.xsd -->
<xs:schema xmlns:my="http://commerce.example.com/payment" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://commerce.example.com/payment"
  elementFormDefault="qualified">

  <xs:element name="payment">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="my:cc" />
          <xs:element ref="my:exp" />
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="as" type="my:paymentAs" />
    </xs:complexType>
  </xs:element>

  <xs:element name="cc" type="my:cc" />
  <xs:element name="exp" type="xsd:gYearMonth" />

  <xs:simpleType name="cc">
    <xs:restriction base="xsd:string">
      <xs:minLength value="12" />
      <xs:maxLength value="19" />
      <xs:pattern value="[0-9]+" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="paymentAs">
    <xs:restriction base="xsd:string">
      <xs:enumeration value="cash" />
      <xs:enumeration value="credit" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>

```

H.2 Editing Hierarchical Bookmarks Using XForms

```

<html xmlns="http://www.w3.org/2002/06/xhtml12" xmlns:xforms="http://www.w3.org/2002/xforms"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:my="http://commerce.example.com/payment"
  xmlns:ev="http://www.w3.org/2001/xml-events" xml:lang="en">
  <head>
    <style type="text/css">
      xforms|input.editField {
        font-weight:bold; font-size:20px; width:500px
      }
      xforms|label.sectionLabel {
        font-weight:bold; color:white; background-color:blue
      }
      xforms|submit {
        font-family: Arial; font-size: 20px; font-style: bold; color: red
      }
    </style>
  </head>

```

```

</style>
<title>Editing Hierarchical Bookmarks In X-Smiles </title>
<xforms:model id="bookmarks" version="1.1">
  <xforms:instance resource="bookmarks.xml" />
  <xforms:submission id="s01" method="post" action="http://examples.com/" />
</xforms:model>
</head>
<body>
  <xforms:repeat nodeset="section" id="repeatSections">
    <xforms:input ref="@name" class="editField">
      <xforms:label class="sectionLabel">Section</xforms:label>
    </xforms:input>
    <!-- BOOKMARK REPEAT START -->
    <xforms:repeat nodeset="bookmark" id="repeatBookmarks">
      <xforms:input ref="@name">
        <xforms:label>Bookmark name</xforms:label>
      </xforms:input>
      <xforms:input ref="@href">
        <xforms:label>URL</xforms:label>
      </xforms:input>
    </xforms:repeat>
  </xforms:repeat>
  <p>
    <!-- INSERT BOOKMARK BUTTON -->
    <xforms:trigger id="insertbutton">
      <xforms:label>Insert bookmark</xforms:label>
      <xforms:insert nodeset="section[index('repeatSections')]/bookmark" at="index('repeatBookmarks')" position="after"
ev:event="DOMActivate" />
    </xforms:trigger>
    <!-- DELETE BOOKMARK BUTTON -->
    <xforms:trigger id="delete">
      <xforms:label>Delete bookmark</xforms:label>
      <xforms:delete nodeset="section[index('repeatSections')]/bookmark" at="index('repeatBookmarks')"
ev:event="DOMActivate" />
    </xforms:trigger>
  </p>
  <p>
    <!-- INSERT SECTION BUTTON -->
    <xforms:trigger id="insertsectionbutton">
      <xforms:label>Insert section</xforms:label>
      <xforms:insert nodeset="section" at="index('repeatSections')" position="after" ev:event="DOMActivate" />
    </xforms:trigger>
    <!-- DELETE SECTION BUTTON -->
    <xforms:trigger id="deletesectionbutton">
      <xforms:label>Delete section</xforms:label>
      <xforms:delete nodeset="section" at="index('repeatSections')" ev:event="DOMActivate" />
    </xforms:trigger>
  </p>
  <!-- SUBMIT BUTTON -->
  <xforms:submit submission="s01">
    <xforms:label>Save</xforms:label>
    <xforms:hint>Click to submit</xforms:hint>
  </xforms:submit>
</body>
</html>

```

Initial instance file bookmarks.xml:

```

<!-- This is the bookmarks.xml file -->
<bookmarks>
  <section name="main">
    <bookmark href="http://www.example.com/xforms.xml" name="Main page" />
  </section>
  <section name="demos">
    <bookmark href="http://www.example.com/demo/images.fo" name="images" />
    <bookmark href="http://www.example.com/demo/xf-ecma.xml" name="ecma" />
    <bookmark href="http://www.example.com/demo/sip.fo" name="sip" />
  </section>
  <section name="XForms">
    <bookmark href="file:///C:/source/xmlevents.xml" name="XML events" />
    <bookmark href="file:///C:/source/model3.xml" name="model3" />
    <bookmark href="file:///C:/source/repeat.fo" name="repeat" />
  </section>
</bookmarks>

```

H.3 Survey Using XForms and SVG

The following example shows one possible way of integrating XForms with [SVG 1.1](#). Note that the complete set of rules for integrating XForms and SVG are not fully specified at the time this specification was published. Future versions of the XForms, SVG, or other W3C specifications might define more complete rules for integrating XForms and SVG which might not be compatible with the example below.

Note that the example below does not use SVG's `switch` and `requiredExtensions` features, which are commonly used in conjunction with `foreignObject`.

```

<!-- <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
      "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" -->
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xforms="http://www.w3.org/2002/xforms" xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:s="http://example.com/survey" width="700px" height="600px" viewBox="0 0 700 600">
  <defs>
    <polygon id="bullet" points="-30,-30, -10,-10, -20,10" fill="#007138" />
    <xforms:model id="form1" schema="surveyschema.xsd">
      <xforms:instance id="instance1">

```

```

<s:survey xmlns="http://example.com/survey">
  <s:drink>none</s:drink>
  <s:espressoPrefs>
    <s:numberPerWeek>0</s:numberPerWeek>
    <s:sugar>0</s:sugar>
    <s:lemon>Always</s:lemon>
  </s:espressoPrefs>
</s:survey>
</xforms:instance>
<xforms:submission id="submit1" method="post" action="http://www.example.org/surveyhandler" />
</xforms:model>
</defs>
<title>Espresso survey</title>
<desc>Sample SVG and XForms - espresso customer survey</desc>
<g>
  <text x="50" y="70" font-size="40" font-family="Arial Black, sans-serif" font-weight="900">Customer Survey: Espresso</text>
  <g font-family="Arial, Helvetica, sans-serif" font-size="18">
    <foreignObject x="80" y="150" width="250" height="40">
      <xforms:select1 appearance="minimal" model="form1" ref="s:drink">
        <xforms:label>
          <g transform="translate(80, 140)">
            <use xlink:href="#bullet" />
            <text>Your usual coffee drink is:</text>
          </g>
        </xforms:label>
        <xforms:item>
          <xforms:label>Rich, dark espresso</xforms:label>
          <xforms:value>espresso</xforms:value>
        </xforms:item>
        <xforms:item>
          <xforms:label>Creamy cappuccino</xforms:label>
          <xforms:value>cappuccino</xforms:value>
        </xforms:item>
        <xforms:item>
          <xforms:label>Long, milky latte</xforms:label>
          <xforms:value>latte</xforms:value>
        </xforms:item>
        <xforms:item>
          <xforms:label>Don't like coffee!</xforms:label>
          <xforms:value>none</xforms:value>
        </xforms:item>
      </xforms:select1>
    </foreignObject>
    <foreignObject x="80" y="240" width="250" height="40">
      <xforms:range model="form1" start="0" end="30" step="5" ref="s:espressoPrefs/s:numberPerWeek">
        <xforms:label>
          <g transform="translate(80, 230)">
            <use xlink:href="#bullet" />
            <text>Shots of espresso per week:</text>
          </g>
        </xforms:label>
      </xforms:range>
    </foreignObject>
    <foreignObject x="80" y="350" width="250" height="40">
      <xforms:select model="form1" ref="s:espressoPrefs/s:sugar">
        <xforms:label>
          <g transform="translate(80, 340)">
            <use xlink:href="#bullet" />
            <text>Sugar?</text>
          </g>
        </xforms:label>
        <xforms:item>
          <xforms:label>Yes</xforms:label>
          <xforms:value>X</xforms:value>
        </xforms:item>
      </xforms:select>
    </foreignObject>
    <foreignObject x="80" y="420" width="250" height="90">
      <xforms:select1 appearance="full" model="form1" ref="s:espressoPrefs/s:lemon">
        <xforms:label>
          <g transform="translate(80, 410)">
            <use xlink:href="#bullet" />
            <text>Lemon?</text>
          </g>
        </xforms:label>
        <xforms:item>
          <xforms:label>Required for the full experience</xforms:label>
          <xforms:value>Always</xforms:value>
        </xforms:item>
        <xforms:item>
          <xforms:label>Whatever</xforms:label>
          <xforms:value>Indifferent</xforms:value>
        </xforms:item>
        <xforms:item>
          <xforms:label>Keep that citrus to yourself</xforms:label>
          <xforms:value>Never</xforms:value>
        </xforms:item>
      </xforms:select1>
    </foreignObject>
  </g>
  <use xlink:href="#bullet" x="101" y="64" transform="scale(7,3)" />
  <foreignObject y="150" x="500" height="60" width="100">
    <xforms:submit model="form1">
      <xforms:label>Send survey</xforms:label>
    </xforms:submit>
  </foreignObject>

```

```
<!-- keep the graphics data out of this example listing -->
<image xlink:href="espresso.svg" x="400" y="230" width="280" height="270" />
</g>
</svg>
```

I Acknowledgements (Non-Normative)

This document was produced with the participation of Forms Working Group participants, including:

- John M. Boyer, IBM (*Chair, Editor*)
- Steven Pemberton, W3C/CWI (*Activity Lead, W3C Team Contact, Chair until 2007*)
- Blake Jones, DAISY Consortium and ViewPlus Technologies
- Ulrich Nicolas Lissé, DreamLab
- Sebastian Schnitzenbaumer, DreamLab (*Co-chair until 2003*)
- Joern Turner, DreamLab
- T. V. Raman, Google
- Keith Wells, IBM
- Charlie Wiecha, IBM
- Nick Van den Bleeken, Inventive Designers n.v.
- Erik Bruchez, Orbeon
- Mark Seaborne, Origo, PicoForms
- Kenneth Sklander, PicoForms
- Susan Borgrink, Progeny Systems
- Rafael Benito Ruiz de Villa, SATEC
- Rogelio Pérez Cano, SATEC
- Lars Oppermann, Sun Microsystems
- Mark Birbeck, Backplane Ltd. (*Invited Expert*)
- Leigh L. Klotz, Jr., Xerox Corporation

J Production Notes (Non-Normative)

This document was encoded in the XMLspec DTD v2.6. The XML sources were transformed using diffspec and xmlspec stylesheets, version 2.6. The XML Schema portion of the Appendix was rendered into HTML with the [xmlverbatim XSLT](#) stylesheet (used with permission). The primary tool used for editing was XMLSpy. The XML was transformed using the XSLT processor in Java 6. The editor(s) use the W3C CVS repository and the W3C IRC server for collaborative authoring.