IBM® Workplace Forms™

Version 2.6.1

Using XForms

Note

Before using this information and the product it supports, read the information in "Notices," on page 95.

First Edition (September 2006)

This edition applies to version 2.6.1 of IBM Workplace Forms and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Introduction

One of the goals of XML is to enable inter-operation of applications by representing data in a standard, human-readable format that is also well suited for processing by computer. XFDL has embraced this standard by incorporating XForms.

XForms standardizes the core business processing model of a web application, but it is designed to be incorporated into host languages that provide extensions as necessary to satisfy diverse additional requirements of web applications.

For example, consider an application that processes purchase orders. A typical PO system would receive a purchase order, then spend significant effort parsing the PO and extracting the data before it began processing. In fact, extracting the data from the PO is often a complicated process that requires a good deal of custom programming.

However, with XForms the PO application can be designed to accept a set of XML data that is defined by a schema. This makes the submission process easier, since the PO system can quickly retrieve the block of XML from the form and begin processing almost immediately. In fact, extracting the data from the form can be as simple as a single line command, which greatly reduces the scope of work necessary for integration.

This document explains what XForms does and provides practical instructions for using it.

Who should read this document

This document is written for form developers who want to use XForms. This document assumes that the reader has a working knowledge of XML and XFDL, as well as some familiarity with XPath and XML Namespaces.

Document conventions

- This guide uses *"xx"* or *"xxx"* in place of the two or three digit version numbers. In particular, these placeholders appear when referring to filenames, folders, and directories that contain version numbers.
- This guide uses <*name*> to identify placeholders identifying nodes in an XPath expression.
- This guide uses ellipses and italics to identify placeholders representing XFDL and XForms elements. For example:

```
...xforms_label...
```

About XForms

XML is a common language for representing data processing models. One of the goals of XML is to enable interoperation of applications by representing data in a standard, human-readable format that is also well suited for processing by computer.

XForms is an XML vocabulary that provides the ability to wrap, or *skin*, XML data with a processing model, including:

- Dynamically recalculated computes that express relationships in data and their properties.
- Static constraints expressed via XML schema constructs.
- Dynamic constraints that are automatically rechecked as data changes.
- Parameters for submitting XML data.
- Event-triggered action sequences that manipulate data in response to user actions.

XForms and XFDL

XFDL is a host language, or skin, for XForms that provides:

- Precision control of layout and form appearance.
- Digital signatures that secure data, presentation, computes, and XForms.
- Offers unique layout and overlap testing that maintains security in sophisticated workflows with multiple signers.
- A rich and extensible declarative compute system that extends to the presentation layer, simplifying management of dynamic user interface behaviors.
- Offline processing via full document save, load, and submission features.
- Internationalization features, including understanding of currency values, and rich accessibility functionality.
- Allows rich text, compressed attachments, and multiple attachments with one user interface control.

When to use XForms

You can use XForms in any of the following scenarios:

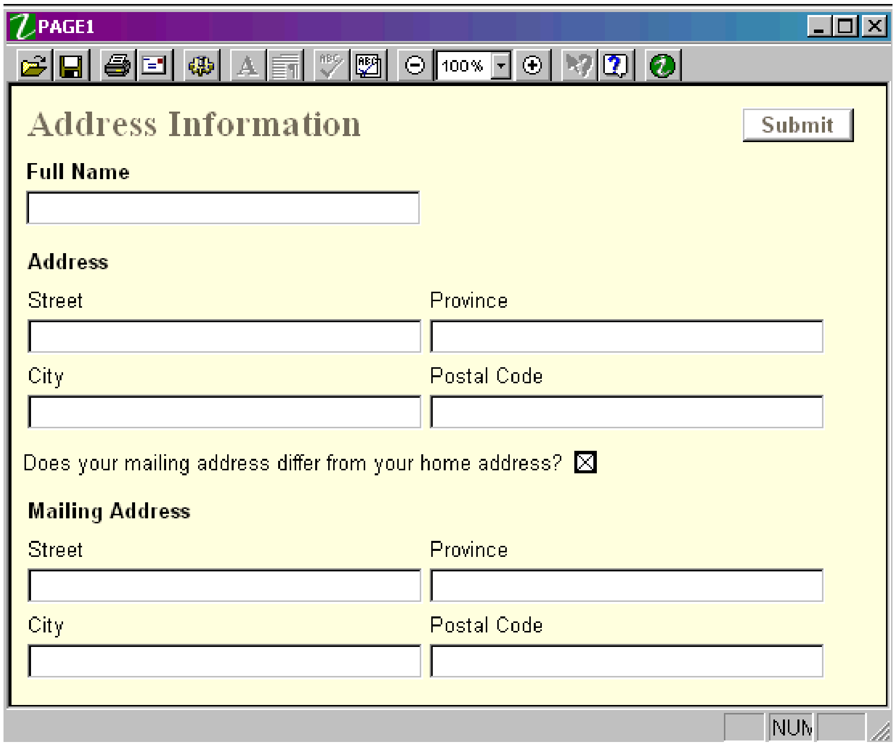
- **XML Applications —** XForms is most useful when integrating eforms with applications that already use XML, especially if those applications already offer XML interfaces. In these cases, you can design forms that will submit the XML data directly to the application, and will not need to program a custom module that extracts the data from the form. Furthermore, you can format the data to match any schema, and validate the data against the schema before submission.
- **Non-XML Applications —** Even if an application does not use XML, you can still benefit from using XForms. The data model simplifies copying information from one page to another, making wizard-style forms easier to create and manage. Furthermore, although custom programming is still required for back-end processing, the data model makes it far easier to extract data from the form.

XForms and the XML data model

You can’t have both an XML Data model and an XForms model in the same form. XForms and the XML Data Model feature are mutually exclusive.

Overview

In this overview, we will explore the presentation layer of the form and how items are displayed. Then we will identify the parts of the XForms model and what they are used for. For the purposes of this demonstration, we will be using the following form:



Address Information

Full Name

Address

Street

Province

City

Postal Code

Does your mailing address differ from your home address? ☒

Mailing Address

Street

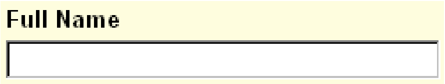
Province

City

Postal Code

Form presentation

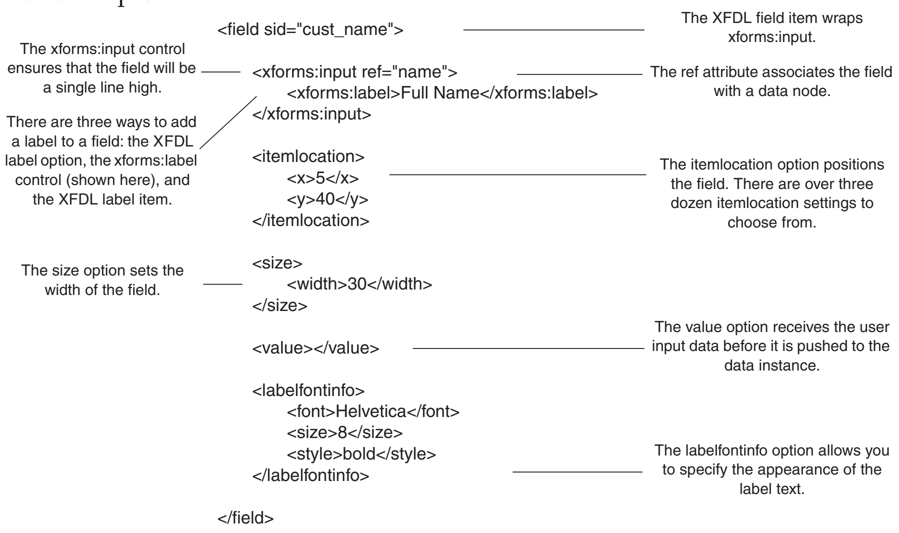
XForms is not a stand alone language. It must be contained inside an XFDL form which acts as a *wrapper* or *skin* that provides detailed presentation information for the form. Furthermore, individual XForms User Interface (UI) elements must be contained by individual XFDL items. For example, consider the **Full Name** field in the example form:



Full Name

To create this field, an XForms *control* called *xforms:input* must be wrapped inside the XFDL *field* item. *XForms:input* determines the basic appearance of the item: a single line field. XFDL code specifies the field's position on the form and it's presentation, such as font and background colors, font type and style, and so on.

For example:



```
<field sid="cust_name">

   <xforms:input ref="name">
      <xforms:label>Full Name</xforms:label>
   </xforms:input>

   <itemlocation>
      <x>5</x>
      <y>40</y>
   </itemlocation>

   <size>
      <width>30</width>
   </size>

   <value></value>

   <labelfontinfo>
      <font>Helvetica</font>
      <size>8</size>
      <style>bold</style>
   </labelfontinfo>

</field>
```

The XFDL field item wraps xforms:input.

The xforms:input control ensures that the field will be a single line high.

The ref attribute associates the field with a data node.

There are three ways to add a label to a field: the XFDL label option, the xforms:label control (shown here), and the XFDL label item.

The itemlocation option positions the field. There are over three dozen itemlocation settings to choose from.

The size option sets the width of the field.

The value option receives the user input data before it is pushed to the data instance.

The labelfontinfo option allows you to specify the appearance of the label text.

In the example above, the XForms UI provides an XForms *control* that ensures that the **Full Name** field always looks like a single-line field regardless of what browser or platform is displaying the form. The UI also performs a number of other tasks, including:

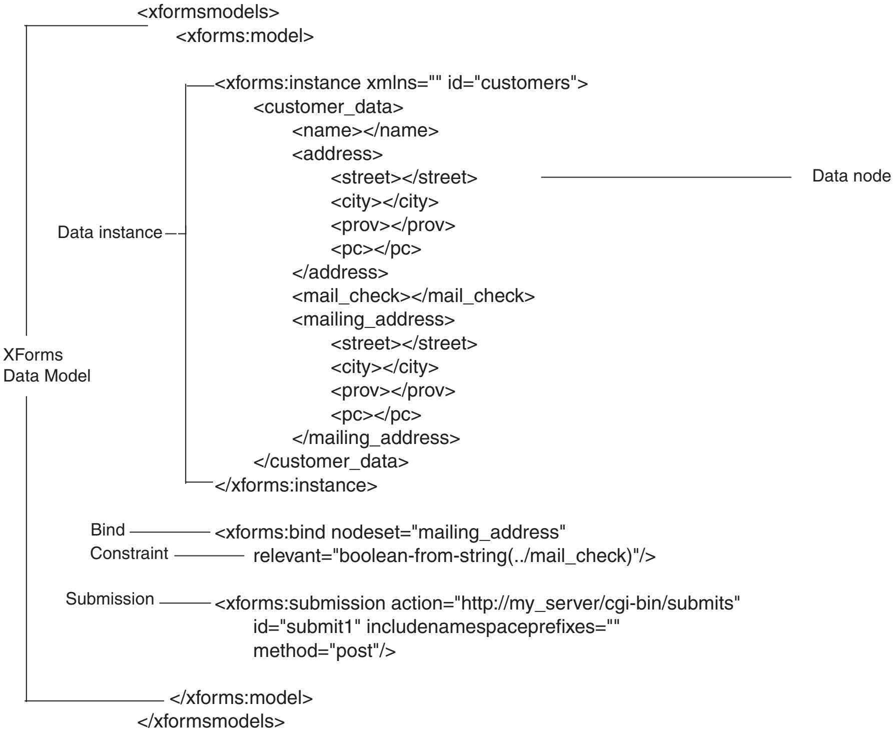
- Automatically accepting instructions from the data model, such as the results of calculations or other properties that effect the form control's appearance.
- Allowing users to update the data instance through the control's association with a data node.
- Providing special grouping and repeating constructs such as tables, groups, and panes.

The XForms model

The XForms model resides at the beginning of a form, inside the form global. It defines the data structure of the form, including:

- XML data instances
- XML schemas
- Binds
- Dynamic constraints
- Submissions rules

The XForms model for our sample form is very simple. It contains one data instance, one bind that determines whether the ″Mailing Address″ section is relevant, and one submission:



```
<xformsmodels>
    <xforms:model>

        <xforms:instance xmlns="" id="customers">
            <customer_data>
                <name></name>
                <address>
                    <street></street>
                    <city></city>
                    <prov></prov>
                    <pc></pc>
                </address>
                <mail_check></mail_check>
                <mailing_address>
                    <street></street>
                    <city></city>
                    <prov></prov>
                    <pc></pc>
                </mailing_address>
            </customer_data>
        </xforms:instance>

        <xforms:bind nodeset="mailing_address"
            relevant="boolean-from-string(../mail_check)"/>

        <xforms:submission action="http://my_server/cgi-bin/submits"
            id="submit1" includenamespaceprefixes=""
            method="post"/>

    </xforms:model>
</xformsmodels>
```

As you can see, the XForms model contains three core parts that work together to create a complete model:

- **Data Instances —** Data instances are arbitrary blocks of XML. A data model may contain any number of data instances, and each instance is normally created to serve a particular purpose.
- **Binds —** Each data instance has associated bindings. Bindings are used to perform special calculations or place limitations on user data. For example, you might want to perform a calculation on a certain element, or specify that user input is mandatory in a certain field.
- **Submission Rules —** Each data instance may have an associated set of *submission rules*. These rules control how a data instance is transmitted when it is submitted for processing. This is an optional feature, and is only necessary when you want to submit the data instance by itself, without the rest of the form. There are many cases in which you may want to submit the entire form, and then retrieve the data instance from the form during processing. This is particularly true when you are using signatures on your forms.

Using XPath

XForms uses XPath to address data nodes in binds, to express constraints, and to specify calculations. Simple XPath expressions resemble file system and directory paths in appearance. But instead of moving through files and folders, XPath expressions move through data nodes. For example, consider the following data instance:

```
<purchaseOrder>
   <products>
      <gadget></gadget>
      ...other nodes...
   </products>
</purchaseOrder>
```

If you wanted to refer to the <gadget> node, you would also need to reference its parent nodes. For example:

```
purchaseOrder/items/gadget
```

When you create your data model, its nodes represent all of the elements, attributes, data, and processing instructions you want to have in the form. Like a file system, these nodes can be represented by a tree. The root element of an instance is the one immediately inside the <xforms:instance> tag. The other elements and attributes form the branches of the tree, while attributes and text values form the leaves.

XPath referencing allows you to associate these nodes or nodesets with items displayed in the form, or with Model Item Properties (MIPs) that calculate values or place limitations on the nodes.

Each node has the following properties:

- **Name** — The name of the node. For example:

  ```
  <gadget>
  ```

- **Value** — The data contained in the node. For example:

  ```
  <gadget>7</gadget>
  ```

  Note: The value may also be empty.

- **Parent** — Every node except the root node has a parent. In the following example, <products> is the parent of <gadget>:

  ```
  <products>
     <gadget></gadget>
  </products>
  ```

- **Children** — Nodes contained within another node. In the following example, <gadget> is the child of <products>:

  ```
  <products>
     <gadget></gadget>
  </products>
  ```

- **Position** — The location of the node relative to all the other nodes. For example:

  ```
  purchaseOrder/items/gadget
  ```

Additionally, some nodes have the following properties:

- **Attributes** — Additional parameters added to the node that extends its functionality. For example:

```
<price currency="USD"></price>
```

- **Namespaces —** Namespace parameters identify a node as being part of a particular namespace. For example, to indicate that the <approvalNumber> node is in the *xmlns:finance="http://example.org/finance"* namespace, you would write:

  ```
  <Finance:approvalNumber>
  ```

Constructing XPath expressions

Just as a relative directory path is relative to a current directory in a file system, an XPath expression is evaluated relative to its current position and environment. This is known as *context*. Factors that can affect the context of an expression include:

- **Context Node —** The current node.
- **Context Position** — An integer indicating the position of the context node in the current node set. Note that the first position is 1, not 0.
- **Context Size** — An integer indicating the number of nodes in the current node set.
- **Variable Bindings —** The mapping between variable names and variable values.
- **Functions** — The library of functions available to an expression. XPath defines a core set of functions that is expanded upon by XForms.
- **Namespace Declaration**s — The mapping between namespace prefixes and namespace URIs.

  Note: The context position is always less than, or equal to the context size.

Simple XPath expressions (such as *purchaseOrder/items/gadget*) are most often used to link a UI element to a node in the data instance. However, you can also construct XPath expressions that yield other results. These include:

- **Nodesets** — A collection of unique nodes within a parent node.
- **Booleans** — True or false.
- **Numbers** — Any floating point number.
- **Strings** — Any sequence of characters.

XPath shortcuts

A large part of any XPath expression is the location path. Location paths determine the node or nodeset over which the rest of the XPath expression operates. As location paths, like directory paths, can be dauntingly long, XPath has provided a number of "shortcuts" that allow you to create less complex-looking expressions.

The following table shows a number of characters that are used as shortcuts:

Shortcuts	**Definition**
.	Selects the context node.
..	Selects the parent node of the context node.
*****	Wild card. Selects any child node.
node()	Indicates any child node. This shortcut is a super-set of *.
next()	Indicates any child text nodes. This shortcut is a sub-set of **node()**.
/	Indicates that a path is absolute when placed at the beginning of a location path. The beginning slash changes the context node to the root element.

*<name>***/<name>**
When a slash follows the name of a node, the first nodes acts as the context of the following node. By default, the following node name is a reference to the children of the context node.

| Indicates a nodeset union.

@ Indicates an attribute node.

Some examples of XPath expressions include:

XPath Expression
Definition

<*name>*
Selects any child nodes called <*name*>.

./<*name***>**
Selects any child nodes called <*name*>.

<*name1>/***<****name2>*
Selects any grand child node called <*name2*> inside any child node called <*name1*>.

.//<*name>*
Selects any nodes called <*name*> in the current sub-tree.

<*name1>* | ***<****name2>*
Selects any child node called <*name1*> or <*name2*>.

*<name>***[1]**
Selects the first child node called <*name*>.

Creating relative location paths

Relative location paths are based on the node that currently has the context. This is the default evaluation scheme used in XForms. For example, consider the following instance:

```
<xforms:model>
   <xforms:instance xmlns="">
      <root>
         <x>
            <y>5</y>
         </x>
         <submitText>Click to submit</submitText>
      </root>
   </xforms:instance>
</xforms:model>
```

Within the body of your form, you may have an *xforms:group* that contains a field with an *xforms:input*. Both of these elements are bound to the data model, as shown:

```
<xforms:group ref="x">
   <field sid="Number">
      <xforms:input ref="y">
         <xforms:label>Number:</xforms:label>
      </xforms:input>
   </field>
</xforms:group>
```

In this case, the *xforms:group* is evaluated relative to the root node of the data model, which in this case is <root>. This links the group to the <x> element. The *xforms:input*, being a child of the *xforms:group* element, inherits its starting point from the *xforms:group* and is evaluated relative to the <x> element. This links the *xforms:input* to the <y> element.

Other XForms elements do not expressly declare which node they are bound to. In these cases, the children of that element inherit the same starting point as the element itself.

For example, consider the following *xforms:group*:

```
<xforms:group ref="x">
   <button sid="Submit">
      <xforms:trigger>
         <xforms:label ref="submittext"/>
      </xforms:trigger>
   </button>
</xforms:group>
```

In this case, the *xforms:trigger* element inherits a starting point of <x> from the *xforms:group*. Since the *xforms:trigger* does not declare a single node binding itself, the *xforms:label* also inherits a starting point of <x> from the *xforms:group*.

Creating absolute location paths

Absolute location paths are preceded with a slash, and begin with the root element of the data instance. For example, consider the following instance:

```
<xforms:model>
   <xforms:instance xmlns="">
      <root>
         <a>
            <b>5</b>
         </a>
         <c>
            <d>10</d>
         </c>
      </root>
   </xforms:instance>
</xforms:model>
```

To create an absolute locations that points to the <d> element, you would write:

```
/root/c/d
```

Additionally, absolute location paths have no inheritances. For example, consider the following *xforms:group* element:

```
<xforms:group ref="a">
   <field sid="Number1">
      <xforms:input ref="b">
         <xforms:label>First Number:</xforms:label>
      </xforms:input>
   </field>
   <field sid="Number2">
      <xforms:input ref="/root/c/d">
         <xforms:label>Second Number:</xforms:label>
      </xforms:input>
   </field>
</xforms:group>
```

In this case, the location path for the first *xforms:input* is evaluated from the <a> element because it inherits this starting location from the *xforms:group*. However, the location path for the second *xforms:input* overrides that inheritance and links to the <d> element.

Using namespaces in element references

References assume that you are working in the empty namespace. To refer to an element that is in a specific namespace, you must include the appropriate namespace prefix in your reference, as shown:

```
prefix:element@prefix:attribute
```

An element might be in a non-default namespace for two reasons. First, the default namespace of the data instance may not be XForms. Second, the element may have a namespace prefix that places it in a non-XForms namespace.

For example, in the following data instance a namespace prefix is used to place some of the elements in a Customer Data (CD) namespace:

```
<xforms:instance xmlns:CD="http://example.org/customerData">
   <CustomerProfile>
      <CD:firstName></CD:firstName>
      <CD:lastName></CD:lastName>
   </CustomerProfile>
</xforms:instance>
```

In this case, to refer to the *firstName* element, you would use the following reference:

```
../customerProfile/CD:firstName
```

Referencing attributes in the data instance

You may need to reference an attribute in the data instance rather than an element. To do this, use the following notation in your reference:

```
element/@attribute
```

For example, consider the following instance:

```
<xforms:instance>
   <LoanRecord>
      <StartDate></StartDate>
              <Borrower></Borrower>
    <Principal currency="USD"></Principal>
    <Duration></Duration>
    <InterestRate></InterestRate>
    <MonthlyPayment></MonthlyPayment>
    <TotalPayout></TotalPayout>
   </LoanRecord>
</xforms:instance>
```

In this case, the *principal* element stores the type of currency required by the loan. To refer to the *currency* attribute, you would use the following reference:

```
<xforms:select1 ref="Principal/@currency">
   ...other select1 options...>
</xforms:select1>
```

Creating an XForms model

Every XForms form contains at least one data model, which in turn contains multiple data instances. As you create an XForms model, it's a good idea to create your data instances one at a time, and to set up the bindings and submission rules for that instance before moving on to the next data instance. This helps to avoid confusion.

To create an XForms model, you must:

- Create the XForms model wrapper.
- Create a data instance.
- Bind the elements of the data instance.
- Create User Interface links.
- Add XForms actions (optional).
- Set up submission rules for the instance (optional).
- Create a submission button for the instance (optional).
- Include a schema (optional).

Creating the XForms wrapper

The XForms model is a block of XML data. It also describes a form's logic, submission rules, and actions. On its own, the XForms model has no graphic component or any way to interact with users. This means that the XForms model must always be contained in a host language or *wrapper*, that provides a presentation layer for the XForms model. In this case, you will use XFDL to create the form content that allows your users to interact with the XForms model. This means you must place your XForms model inside an XFDL form.

To embed an XForms model inside an XFDL form, you must:

- Define the XForms namespaces.
- Declare the XForms model.

Defining the XForms namespace

Namespaces are used to identify different information in the same XML document. Information is usually categorized by its purpose. For example, the XFDL namespace contains information that describes electronic forms.

You can think of a namespace as being similar to the name of a language, with each language intended for a different reader. For example, a book might contain English for english readers and Russian for russian readers.

To define a namespace, you must use the *xmlns* attribute. This attribute assigns the unique URI of the namespace to a prefix, as shown:

```
xmlns:prefix="namespace URI"
```

This assignment allows a simple prefix to specify the element's namespace URI. Your prefixes should succinctly describe the namespace, so that it is easy to remember and recognize. By convention, the XForms prefix is *xforms.*The namespace URI can be any unique URL that describes the namespace. For example, the XForms namespace is defined by the following URI:

```
http://www.w3.org/2002/xforms
```

Substituting these values, you get the following namespace declaration:

```
xmlns:xforms="http://www.w3.org/2002/xforms"
```

This attribute should be defined on the document element node of the XFDL form. For example:

```
 <XFDL xmlns:xforms="http://www.w3.org/2002/xforms">
```

Using namespaces

Because you are using XFDL as your XForms host language, your default namespace should be XFDL. While most of your form elements will probably be in the default namespace, there may be times when you want to create a namespace that you use selectively. For example, you might have a data instance that should be in your company's default namespace, except for two elements that should be in the Human Resources namespace. In this case, you would assign the Human Resources namespace to a prefix, and then use that prefix to tag specific data elements.

When defining other namespaces to use, it is best to declare them at the beginning of the form. This ensures that they are available for use throughout the form. You use the *xmlns* attribute to assign the unique URI for the namespace to a namespace prefix, as shown:

```
xmlns:prefix="namespace URI"
```

For example, the following tag creates an *hr* prefix for a Human Resources namespace:

```
xmlns:hr="http://www.acme.com/HR"
```

You can now add the prefix to any tag within a data instance to indicate that the tag belongs to the Human Resources namespace, as shown:

```
prefix:tag
```

For example, if you wanted the first name and last name in a data instance to belong to the Human Resources namespace, you would write:

```
<purchaseOrderData>
   <hr:firstName></hr:firstName>
   <hr:lastName></hr:lastName>
   <address></address>
</purchaseOrderData>
```

In this case, both the first and last name are in the Human Resources namespace, but the address is not since it has no prefix. Also, notice that each closing tag must also include the prefix.

Declaring the XForms data model

The XForms model is declared as an option in the global item of the forms global page. It begins with the <xformsmodels> tag, as shown:

```
<globalpage sid="global">
   <global sid="global">
      <xformsmodels>
         ...
      </xformsmodels>
   </global>
</globalpage>
```

Once you have created the XForms container, you can place your data models inside. Each model is enclosed by an <xforms:model> tag. You can add as many data models as necessary, although they must all be placed within the same <xformsmodels> container. For example:

```
<globalpage sid="global">
   <global sid="global">
      <xformsmodels>
         <xforms:model>
            ...data model 1...
         </xforms:model>
         <xforms:model>
            ...data model 2...
         </xforms:model>
      </xformsmodels>
   </global>
</globalpage>
```

Naming a data model

While the first model in the form is the default model and doesn't require a name, any additional XForms models must have a specific name. You can give each data model a unique name by using the *id* attribute. The *id* attribute is added to the <xforms:model> tag of your data model, and follows this format:

```
id="name"
```

For example, if you wanted to call the second model in your form "model_2", you would use the following tag to begin the model:

```
<xforms:model id="model_2">
```

Creating data instances

Each data instance is created within the <xforms:model> tag, and begins with an <xforms:instance> tag, as shown:

```
<xforms:model>
   <xforms:instance>
      ...data instance 1...
   </xforms:instance>
   <xforms:instance>
      ...data instance 2...
   </xforms:instance>
</xforms:model>
```

Each <xforms:instance> tag contains a data instance. This can be any XML data you like, but must be well-formed XML, meaning that it must have a single root element. You should give the root element a meaningful name that reflects the content of the instance. For example, you might use a <LoanRecord> element to begin an instance that contains data for a loan application, as shown:

```
<xforms:instance>
   <LoanRecord>
      ...loan data...
   </LoanRecord>
<xforms:instance>
```

Your data instance can contain any valid XML. For example, your loan data might include the start and end date of the loan, the name of the borrower, the amount of the loan, and so on. In this case, your data instance might look like this:

```
<xforms:instance>
   <LoanRecord>
      <StartDate></StartDate>
            <Borrower></Borrower>
```

```
    <Principal></Principal>
    <Duration></Duration>
    <InterestRate></InterestRate>
    <MonthlyPayment></MonthlyPayment>
    <TotalPayout></TotalPayout>
   </LoanRecord>
</xforms:instance>
```

Instance data is not processed by XFDL parsers, such as Workplace Forms Viewer and Workplace Forms Designer, and can follow any format necessary. This gives you the freedom to create XForms models that match defined schemas or other formats.

Naming a data instance

If you have more than one data instance in your form, you must give a unique name to each instance. You can do this by adding an *id* attribute to the <xforms:instance> tag of your data instance. The *id* attribute follows this format:

```
id="name"
```

For example, if you wanted give the name *loan* to the LoanRecord data instance, you would use the following tag to begin the data instance:

```
<xforms:instance id="loan">
```

Defining the default namespace

If you want your default namespace for the instance to differ from the default instance for your form, you must add an *xmlns* attribute to the <xforms:instance> tag. The *xmlns* attribute is assigned the URI that defines the namespace, as shown:

```
xmlns="namespace URI"
```

For example, if you wanted to place an instance in your company's Human Resources namespace, the <xforms:instance> tag of our customer data might look like this:

```
<xforms:instance xmlns="http://www.mycompany.com/namespaces/HR">
```

When you set the default namespace for an <xforms:instance> tag, both the opening node and all children of that node are placed in that namespace. However, there may be times when some of that node's children should belong to a different namespace. If so, you must identify that namespace by adding the prefix of a defined namespace to the node's tag, as shown:

```
prefix:tag
```

For example, the following instance is in the XForms namespace, but the <LenderID> node is in the HR namespace:

```
<xforms:instance>
   <LoanRecord>
      <HR:LenderID></HR:LenderID>
      <StartDate></StartDate>
             <Borrower>         </Borrower>
    <Principal></Principal>
    <Duration></Duration>
    <InterestRate></InterestRate>
    <MonthlyPayment></MonthlyPayment>
    <TotalPayout></TotalPayout>
   </LoanRecord>
</xforms:instance>
```

Setting the properties of a data instance

Once you have created a data instance, you need to set the properties of its nodes. This is done using the <xforms:bind> tag. These binds are used to perform special calculations or place limitations on user data. For example, you might want to perform a calculation on a certain node or ensure that supplying certain data is mandatory.

Binds are contained within the XForms data model. They are placed below the data instances. For example:

```
<xforms:model>
   <xforms:instance>
      ...data instance 1...
   </xforms:instance>
   <xforms:instance>
      ...data instance 2...
   </xforms:instance>
   <xforms:bind/>
   <xforms:bind/>

</xforms:model>
```

You create a bind by adding properties to the <xforms:bind> start tag. For example:

```
<xforms:bind ...bind information.../>
```

These properties allow you to:

- Name the bind. (optional)
- Determine the node or nodeset that is affected by the bind. (mandatory)
- Describe the way the bind effects the element. (mandatory)

These attributes may be added to the <xforms:bind> start tag in any order. However, the name and nodeset attributes are typically listed first, as each bind will never have more than one name and nodeset, while it may have a number of properties that effect the bind. For example:

```
<xforms:bind ...name... ...nodeset... ...property_1... ...property_2.../>
```

Naming binds

The *id* attribute allows you to give a globally unique name to your bind. This attribute is optional, but necessary if you want to refer to the bind elsewhere in your form. The format of the *id* attribute is:

```
id="bind name"
```

In the following code sample, the bind is named “TotalPayoutBind”:

```
<xforms:bind id="TotalPayoutBind" ...nodeset... ...property_1...
   ...property_2.../>
```

Note: IDs are only allowed on parent binds. They are not supported on binds nested that are inside another bind.

Determining the affected node

A *nodeset* identifies which element is affected by the bind and must refer to a node in a data instance. Every bind is associated with a nodeset, either directly, or in the case of nested binds, through inheritance from a parent bind. The format of the *nodeset* attribute is:

```
nodeset="path_to_node"
```

You identify these nodesets using XPath expressions that provide the location path of the node. For example, the following bind specifies the <LoanTotal> node in the default instance:

```
<xforms:bind ...id... nodeset="LoanTotal" ...property1...
      ...property_2.../>
```

If necessary, you can create multiple binds for a single nodeset.

Assigning properties

Every bind contains one or more model item property (MIPs). A MIP describes the way the bind modifies its associated node. These modifications include determining the value of nodes, their validity, or relevancy.

The format of a MIP is:

```
property_name="property_details"
```

In a loan application, you may want to ensure that the field showing the total of the loan is only visible if the <MonthlyPayment> node has a value greater than 0, you would have to use the *relevant* MIP. The *relevant* MIP determines whether a node is relevant and needs to be displayed. For example:

```
<xforms:bind ...id... ...<LoanTotal nodeset>...
   relevant="../MonthlyPayment &gt; 0"/>
```

There are six MIPs:

- Calculate
- Constraint
- Readonly
- Relevant
- Required
- Type

Calculate: The *calculate* property defines a calculation that determines the value of the associated node. It allows you to add mathematical formulas and computed logic to your data instance. *Calculate* follows this format:

```
calculate="<calculation>"
```

The essential elements of the calculation are XPath expressions combined with normal mathematical expressions. The following are simple examples of a *calculate*:

```
calculate="<XPath_Expression> + <XPath_Expression>"
calculate="<XPath_Expression> * <XPath_Expression> + 10"
calculate="<XPath_Expression> * 10"
```

As discussed in ″Using XPath″ , an XPath expression provides the location of the node being modified. Thus, if you wanted to multiply the value of an <amount> node by .1, you would write:

```
calculate="amount * 0.1"
```

Constraint: The *constraint* property determines whether an associated node is valid. For example, the *constraint* property could ensure that a node was only valid if its value was greater than zero. In other words, if that node had a value greater than zero, its validity equals *true*. If the node had a value of zero or less, its validity equals *false*. For instance, if you wanted to ensure that a field would contain a number higher than 0 but less than 10, you could use *constraint* to prevent the form from accepting a value that was outside of that range.

Constraint follows this format:

```
constraint="<XPath_Expression> <operator> <XPath_Expression>"
```

You can use both relational and logical operators. Relational operators are character sets that describe how one thing relates to another. For example, the greater than and less than signs are relational operators. Logical operators allow you to create more complex computes with logical or and logical and.

The following table lists all the operators you may use:

Relational Operator	**Definition**		**Logical Operator**	**Definition**
>	greater than		and	logical AND
&lt;	less than		or	logical OR
&lt;=	less than or equal to		**Note:** For logical NOT, use the XForms not() function.	
>=	greater than or equal to			
=	equal to			
!=	not equal to			

Note: You should not use the less than character (<) in your binds. Since it is used to signal the beginning of the bind tag, using it inside your bind will cause your bind to fail. Instead, replace this symbol with a character reference. As noted in the table above, the less than sign can be represented with &lt;.

The *constraint* property uses relational operators to impose limits on a data node. For example, if you wanted to ensure that the value of a node was not equal to 0, you would write:

```
constraint=". != 0"
```

You can also relate the bound node to another node. For example, the following constraint ensures that the value of the associated node must be greater than the value of the <minimum> node:

```
constraint=". > minimum"
```

You can use the logical operators to add additional constraints to the attribute. For example, you could limit the value of the associated node so that it must be greater than the <minimum> node, but less than the <maximum> node:

```
constraint=". &gt; minimum and . &lt; maximum"
```

The result of a *constraint* parameter always either true() or false(). In other words, the value of the node either conforms to the parameter's limitations, or it does not. The default value is true.

Readonly: The *readonly* property determines whether the data in the associated node can be changed. It follows this format:

```
readonly="<XPath_Expression with a result of either true or false>"
```

Readonly accepts any XPath expression as its setting, but the result is always converted to either **true()** or **false()**. Note that if you want to directly declare the settings as true or false, you must express these states as though they were functions, so that the parser doesn't interpret "true" or "false" as a string rather than a limitation on the node. For example, if you want to directly declare a node to be readonly, you would write:

```
readonly="true()"
```

The default value of *readonly* is false(), as most nodes will accept input from the user. However, if the bind includes a *calculate* property, the node automatically has a *readonly* of true(), as the value of the node will be based on a calculation, and not directly on input from the user. Furthermore, if a node is set to *readonly*, then all of its child nodes automatically inherit the *readonly* setting.

Relevant: The *relevant* property determines whether a node is displayed to the user or included in XForms submissions. *Relevant* follows this format:

```
relevant="<XPath_Expression with a result of either true or false>"
```

Relevant accepts any XPath expression as its setting, but the result is always converted to either **true()** or **false()**. Note that if you want to directly declare the settings as true or false, you must express these states as though they were functions, so that the parser doesn't interpret "true" or "false" as a string rather than a limitation on the node. For example, if you want a node to be mandatory, you would write:

```
required="true()"
```

Relevant is very useful for computing visible and non-visible items on a form. For example, the following expression ensures that the value of the associated form items are displayed only if the credit option is selected from the payment group:

```
relevant="../payment = 'credit'"
```

The default value of *relevant* is true. However, all children of the node inherit its setting, so if a node is not relevant, then its children will not be relevant either, and will not be displayed to the user.

Required: The *required* property determines whether a node requires mandatory user input. It follows this format:

```
required="<XPath_Expression with a result of either true or false>"
```

Required accepts any XPath expression as its setting, but the result is always converted to either **true()** or **false()**. Note that if you want to directly declare the settings as true or false, you must express these states as though they were functions, so that the parser doesn't interpret "true" or "false" as a string rather than a limitation on the node. For example, if you want a node to be mandatory, you would write:

```
required="true()"
```

The default value of *required* is false. Note that if a parent node has a *required* of false(), child nodes must also have a *required* of false(). However, if a parent node has a *required* of true, its children may have a *required* of either true() or false(). Furthermore, if the node does not require user input, but a linked UI element does, then input is still required.

Note: XForms submissions will not submit if a relevant data node is required but contains no data.

Type: The *type* property sets the data node to be a particular data type. It is written as:

```
type="<namespace>:<qualified_name>"
```

If the data type set for the node conflicts with the data type set for a linked User Interface element, then the validity of the data is determined by considering both settings. If the data is invalid for either setting, then the data is considered invalid overall.

The available data types are defined by the enclosed XML schema. For example, the schema could define:

- xsd:boolean
- xsd:integer
- xsd:date
- xsd:double

> **Note:** Any type defined by an XML schema associated with the xforms:model can be used.

For example, if you wanted to define a node as a boolean data type, you would write:

```
type="xsd:boolean"
```

The default value is *string*.

Referencing an attribute in the data instance

As discussed in "Using XPath", you may need to reference an attribute in the data instance rather than an element. To do this, use the following notation in your reference:

```
element@attribute
```

For example, consider the following instance:

```
<xforms:instance>
   <LoanRecord>
      <StartDate></StartDate>
             <Borrower></Borrower>
    <Principal currency="USD"></Principal>
    <Duration></Duration>
    <InterestRate></InterestRate>
    <MonthlyPayment></MonthlyPayment>
    <TotalPayout></TotalPayout>
   </LoanRecord>
</xforms:instance>
```

In this case, the *principal* element stores the type of currency required by the loan. To refer to the *currency* attribute, you would use the following reference:

```
<xforms:bind nodeset="Principal/@currency" ...MIP.../>
```

Creating user interface links

User Interface (UI) links connect elements in the user interface to elements in the data instance, linking your data model to the items your users see on the screen. This causes information entered by the user to be copied to your data model. For example, you might link the *lastName* element in your data instance to the *lastName.value* option in your form description.

You place UI links inside the XFDL items that you want to link to the data model, as shown:

```
<field sid="LastName">
   ...UI Link...
</field>
```

UI links are defined by *form controls*. Essentially, form controls are like a special kind of option, except instead of being XFDL options, they are XForms options. These form controls define the appearance of items and the types of content they display. Their format is:

```
<field sid="LastName">
    <...form control...></...form control...>
</field>
```

Form controls are linked to specific data nodes in the XForms model using *XPath references*. XPath references are attributes of form controls, as shown:

```
<field sid="LastName">
   <...form control... ...XPath Ref...></...form control...>
</field>
```

Most form controls may be modified by additional attributes. These attributes usually describe the type of data accepted by the form control or data node, or modify how that data is used. These attributes are added to the form control's opening tag. For example:

```
<field sid="LastName">
   <...form control... ...XPath Ref... ...attributes...>
   </...form control...>
</field>
```

The following sections describes how to create some of the most frequently used XFDL items with XForms controls. Note that items that do not interact with the data model (and thus do not require XForms controls) are not detailed here. For detailed information about non-XForms XFDL, see the *Workplace Forms XFDL Specification* document.

Creating buttons

Generally, you use buttons to allow users to trigger actions, such as sending submissions, uploading attachments, saving forms, and so on. However, not all of these button types need to be linked to the data node. For example, users can save forms or go to the next page without needing to exchange information with the data instance. Thus not all XFDL button types have XForms counterparts.

You can use XForms to create the following types of buttons:

- Submit buttons
- Attachment buttons
- Trigger buttons
- Insert buttons
- Delete buttons

Creating a submit button

Submit buttons submit instance data to a processing server. Clicking the button triggers the submission action. Submission rules in the data model and filters in the submission button determine what data is sent to the server.

You should not create a submit button until you have created submission rules that determine what data should be sent. For detailed information on creating submission rules and submit buttons, see “Creating submission rules for an instance” on page 82 and “Creating a submission button” on page 87.

Creating an attachment button

Attachment buttons allow your users to add, view, or remove files from your forms. However, only a few of these activities are performed using XForms. This means that some attachment buttons can use XForms, while others must be created using XFDL options. The following table shows which method you can use for the type of attachment button you want:

Attachment Button	User Interface Link	XFDL Options
Attach Single File	Yes	Yes
Attach Multiple Files	No	Yes
Display Single File	No	Yes
Display Image File	Yes	Yes
Display Multiple Files	No	Yes
Extract Single File	No	Yes
Extract Multiple Files	No	Yes
Remove Single File	No	Yes
Remove Multiple Files	No	Yes

Attachment buttons only need to include user interface links if you want to enclose a file inside an instance node or to display a file that is already enclosed inside an instance node. If the file does not need to be in the data instance, we recommend that you use the appropriate XFDL options to manipulate attachments.

Creating an attach single file button

If you want users to be able to enclose a file inside an instance node, you must create an Attach Single File button. This is done using the *upload* control. The *upload* control allows users to select a file to be contained inside the instance node.

To create an Attach Single File button, you must:

- Create the instance that will contain the file information.
- Create the XFDL button.
- Add the *upload* control.
- Add its label control.
- Add the child elements of the *upload* control.

Create the instance

The data instance that will contain the attached file must include the following nodes:

- **Data node —** The node that will contain the actual file you are attaching. This node may have any name, but it should accurately describe its function. For example, **imagedata** or **attachedfile**.
- **filename —** The node that will contain the actual name of the file. This node is optional, but if you include it, it must always be called **filename**.

- **mediatype —** The node that describes the file's mediatype. This node must always be called **mediatype**.

The following example shows an instance that is ready to attach an image file:

```
<xforms:instance xmlns="" id="attachedFile">
      <data>
         <imagedata></imagedata>
         <filename></filename>
         <mediatype></mediatype>
      </data>
</xforms:instance>
```

Create the XFDL button

The attachment button is written as a normal XFDL *button* item. Using XFDL options, you can modify the button's appearance and position in any way you want. However, you should not give the button a *type* option, since it will be overridden by the *upload* control.

For example, you might create the following button:

```
<button sid="uploadButton">
   <itemlocation>
      <x>0</x>
      <y>23</y>
      <width>720</width>
      <height>500</height>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Add the upload control

The *upload* control links the button to the XForms data model, so that the file is copied to the specified data instance. It is placed inside the attachment button wrapper.

The format of the *upload* control is:

```
<xforms:upload XPath_Ref mediatype>
   ...label control...
</xforms:upload>
```

Upload has four attributes:

1. **ref —** The path to the data node. This attribute links the *upload* control to the node that will contain the file. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *upload* control is associated with the <imagedata> node inside the <enclosedFile> instance, then the XPath reference would be:

   ```
   ref="instance('enclosedFile')/imagedata"
   ```

2. **mediatype —** Allows you to set the type of file you want to attach. The format of *mediatype* is:

   ```
   mediatype="general_filetype/specific_filetype"
   ```

   For example, if you wanted your users to only upload JPEG images, you would write:

```
mediatype="image/jpeg"
```

You may use an asterisk as a file type wildcard, which would allow users to upload files of any type. For example:

```
mediatype="*/*"
```

3. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id is ″model2″, then you would write:

   ```
   model="model2"
   ```

4. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the button. *Upload* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id is ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

The *upload* control is nested inside the *button*, as shown:

```
<button sid="AttachImage">
   <xforms:upload>
      ...xforms:label...
   </xforms:upload>
   ...XFDL options...
</button>
```

Add the label control

The *label* control is a mandatory element of the *upload* control. It provides text for the button. However, if you prefer use to use the XFDL value option to label the button, the content of *label* may be left blank. If the *label* control has a value, it overrides the XFDL *value* option.

The format of the *label* control is:

```
<xforms:label>button_text</xforms:label>
```

You can use *label* to set the text that the button displays, such as Attach Image. For example:

```
<xforms:label>Attach Image</xforms:label>
```

The *label* control is nested inside the *upload* control, as shown:

```
<xforms:upload>
   <xforms:label>Attach Image</xforms:label>
</xforms:upload>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="AttachImage">
   <xforms:upload>
       <xforms:label>Attach Image</xforms:label>
   </xforms:upload>
   ...XFDL options...
</button>
```

If you want to modify the appearance of the *label* text, you can use the standard XFDL font options. They are placed inside the button item, but outside the *upload* form control. For example:

```
<button sid="UploadButton">
    <xforms:upload
      ref="instance('enclosedFile')/imagedata"
      mediatype="image/jpeg">
      <xforms:label>Attach Image</xforms:label>
   </xforms:upload>
   <fontcolor>blue</fontcolor>
   <fontinfo>
      <font>Helvetica</font>
      <size>8</size>
      <effects>bold</effects>
   </fontinfo>
</button>
```

Add the child elements of the upload control

The *upload* control has two child elements which must also link to elements in the data model:

1. **filename —** Links the image file's name to a specific data node. (optional)
2. **mediatype —** Links the file's mediatype to a specific data node.

These elements are written as:

```
<xforms:filename XPath_Ref></xforms:filename>
<xforms:mediatype XPath_Ref></xforms:mediatype>
```

Earlier in our example, we called these nodes <filename> and <mediatype>, so we would write:

```
<xforms:filename ref="filename"></xforms:filename>
<xforms:mediatype ref="mediatype"></xforms:mediatype>
```

Filename and mediatype are always nested inside the upload control, as shown:

```
<button sid="UploadButton">
   <xforms:upload
      ref="instance('attachedFile')/imagedata"
      mediatype="image/jpeg">
      <xforms:label>Attach Image</xforms:label>
      <xforms:filename ref="filename">></xforms:filename>
      <xforms:mediatype ref="mediatype"></xforms:mediatype>
   </xforms:upload>
   ...XFDL options...
</button>
```

Example of a completed attach single file button

The following code shows a sample of a button that uploads user files to the data instance:

```
<button sid="UploadButton">
   <xforms:upload
      ref="instance('attachedFile')/imagedata"
      mediatype="image/jpeg">
      <xforms:label>Attach Image</xforms:label>
   </xforms:upload>
   <xforms:filename ref="filename"></xforms:filename>
   <xforms:mediatype ref="mediatype"></xforms:mediatype>
   <fontcolor>white</fontcolor>
   <fontinfo>
      <font>Helvetica</font>
      <size>8</size>
      <effects>bold</effects>
   </fontinfo>
   <itemlocation>
```

```
         <x>0</x>
         <y>23</y>
      </itemlocation>
      <size>
         <width>20</width>
         <height>10</height>
      </size>
      <value></value>
   </button>
```

Creating a trigger button

Trigger buttons activate actions. In fact, you could consider any button to be a trigger, since the purpose of a button is to allow users to activate the action it contains. However, in this case a Trigger button specifically refers to buttons that contain the *trigger* control.

For the purposes of this example, this Trigger button will activate a *toggle* action. However, Trigger buttons are not limited to activating *toggle* actions; they can activate any XForms action, such as *insert* and *delete*. In this example, our Trigger button will be used to activate conditional items inside *switch* and *case* controls.

Note: These instructions assume that the case and switch controls affected by the Trigger button have already been created. For more information, see "Creating conditional items" on page 57.

To create a trigger button, you must:

- Create an XFDL button.
- Add the *trigger* control.
- Add the *label* control.
- Add the *toggle* action.

Create the XFDL button

The Trigger button is written as a normal XFDL *button* item. Using XFDL options, you can modify the button's appearance and position any way you want. This includes determining size of the button and the appearance of its text.

For example, you might create the following button:

```
   <button sid="SwitchToSpanish">
      <itemlocation>
         <x>0</x>
         <y>23</y>
      </itemlocation>
      <size>
         <width>20</width>
         <height>10</height>
      </size>
      <value></value>
   </button>
```

Add the trigger control

The *trigger* control allows the user to trigger actions in the form. In this case, *trigger* is used to activate the *toggle* control. *Toggle* selects one possible *case* control from the possible cases in a *switch* group.

The format of the *trigger* control is:

```
   <xforms:trigger></xforms:trigger>
```

The *trigger* control is nested inside the Toggle button item:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the label control

The *label* control is a mandatory element of the *trigger* control. However, if you prefer use to use the XFDL value option to label the button, the content of *label* may be left blank. If the *label* control has a value, it overrides the XFDL *value* option.

The format of the *label* control is:

```
<xforms:label>button_text</xforms:label>
```

You can use *label* to set the text that the button displays, such as Spanish. For example:

```
<xforms:label>Switch to Spanish</xforms:label>
```

The *label* control is nested inside the *trigger* control. For example:

```
<xforms:trigger>
   <xforms:label>Switch to Spanish</xforms:label>
 </xforms:trigger>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
      <xforms:label>Switch to Spanish</xforms:label>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the toggle action

The *toggle* action selects one possible case from all the possible cases contained in a *switch* group. In other words, *toggle* ensures that the selected case is turned on, while the other cases are turned off.

The format of *toggle* is:

```
<xforms:toggle attributes></xforms:toggle>
```

Toggle has two attributes:

- **event** — Indicates the type of event that is triggered. This attribute is mandatory and is written as:

  ```
  ev:event="event"
  ```

  In the case of *toggle*, the event is usually **DOMActivate**.
- **case** — Indicates which case is activated when the user clicks the Toggle button. This attribute is only mandatory if your Trigger button activates conditional items. The format of *case* is:

  ```
  case="name"
  ```

  This attribute must match the *id* of the *case* control you want to select. For example, if you wanted a case with an *id* of "Spanish" to be selected, you would write:

```
case="Spanish"
```

The *toggle* action is nested inside the *trigger* control. For example:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
      <xforms:toggle
         ev:event="DOMActivate"
         case="Spanish">
      </xforms:toggle>
      <xforms:label>Switch to Spanish</xforms:label>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Example of a completed trigger button

The following code shows a sample of a button that switches the language displayed by the form:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
      <xforms:toggle
         ev:event="DOMActivate"
         case="Spanish">
      </xforms:toggle>
      <xforms:label>Switch to Spanish</xforms:label>
   </xforms:trigger>
   <fontcolor>white</fontcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Creating an insert button

Insert buttons allow users to dynamically add new items to groups and panes and rows to tables. This is done using a *trigger* control to activate an XForms *insert* action. *Insert* determines which items are inserted, where the items are inserted, and which event is activated.

For the purposes of our example, we will create an Insert Row button.

To create an insert row button, you must:

- Create an XFDL button.
- Add the *trigger* control.
- Add the *label* control.
- Add the *insert* action.

Create the XFDL button

The Insert Row button is written as a normal XFDL *button* item. Using XFDL options, you can modify the button's appearance and position any way you want. This includes determining the size of the button and the appearance of its text.

For example, you might create the following button:

```
<button sid="InsertRow">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Add the trigger control

The *trigger* control allows the user to trigger actions in the form. In this case, *trigger* is used to activate the *insert* action.

The format of the *trigger* control is:

```
<xforms:trigger> ...xforms_label... </xforms:trigger>
```

The trigger control is nested inside the InsertRow button item:

```
<button sid="InsertRow">
   <xforms:trigger>
      ...xforms_label...
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the label control

The *label* control is a mandatory element of the *trigger* control. It provides text for the button. However, if you prefer use to use the XFDL value option to label the button, the content of *label* may be left blank. If the *label* control has a value, it overrides the XFDL *value* option.

The format of the *label* control is:

```
<xforms:label>button_text</xforms:label>
```

You can use *label* to set the text that the button displays, such as Attach Image. For example:

```
<xforms:label>Insert Row</xforms:label>
```

The *label* control is nested inside the *trigger* control, as shown:

```
<xforms:trigger>
   <xforms:label>Insert Row</xforms:label>
</xforms:trigger>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="InsertRow">
   <xforms:trigger>
      <xforms:label>Insert Row</xforms:label>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the insert action

The *insert* action places the new row in the form. Its attributes determine which items are inserted, where they are inserted, and which event is activated.

The format of the *insert* action is:

```
<xforms:insert ...attributes... />
```

Insert is modified by four attributes:

1. **at** — Determines where the new items will be inserted. This attribute usually refers to an XFDL group-type item, such as *pane, table,* or *group*. The format of *at* is:
   ```
   at="index('item_sid')"
   ```
2. **position** — Indicates whether the inserted node is added before or after the anchor (referenced) node. The format of *position* is:
   ```
   position="before_or_after"
   ```
3. **nodeset** — Indicates the node or nodeset to be inserted. The format of *nodeset* is:
   ```
   nodeset="node"
   ```
4. **event** — Indicates the type of event that is triggered. In the case of *insert*, it is always:
   ```
   ev:event="DOMActivate"
   ```

An example of an *insert* action with all its attributes is:

```
<xforms:insert
   at="index('customer_table')"
   position="after"
   nodeset="customer"
   ev:event="DOMActivate"/>
```

Insert is placed inside of the *trigger* control that activates the action. For example:

```
<xforms:trigger>
   <xforms:label>Insert Row</xforms:label>
   <xforms:insert
      at="index('customer_table')"
      position="after"
      nodeset="customer"
      ev:event="DOMActivate"/>
</xforms:trigger>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="InsertRow">
   <xforms:trigger>
      <xforms:label>Insert Row</xforms:label>
      <xforms:insert
      at="index('customer_table')"
      position="after"
      nodeset="customer"
      ev:event="DOMActivate">
   </xforms:insert>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Example of a completed insert row button

The following code shows a sample of a button that adds an additional row to a table:

```
<button sid="InsertRow">
   <xforms:trigger>
      <xforms:label>Insert Row</xforms:label>
      <xforms:insert
      at="index('customer_table')"
```

```
      position="after"
      nodeset="customer"
      ev:event="DOMActivate"/>
   </xforms:trigger>
   <fontcolor>blue</fontcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Creating a delete button

Delete buttons allow users to dynamically remove unnecessary items from groups or panes and rows from tables. This is done using a *trigger* control to activate an XForms *delete* action. *Delete* determines which items are deleted, where they items are positioned in the table, and which event is activated.

For the purposes of our example, we will create a Delete Row button.

To create a Delete Row button, you must:

- Create an XFDL button
- Add the *trigger* control.
- Add the *label* control.
- Add the *delete* action.

Create the XFDL button

The Delete Row button is written as a normal XFDL *button* item. Using XFDL options, you can modify the button's appearance and position any way you want. This includes determining size of the button and the appearance of its text.

For example, you might create the following button:

```
<button sid="DeleteRow">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Add the trigger control

The *trigger* control allows the user to trigger actions in the form. In this case, *trigger* is used to activate the *delete* action.

The format of the *trigger* control is:

```
<xforms:trigger>...xforms_label...</xforms:trigger>
```

The trigger control is nested inside the DeleteRow button item:

```
<button sid="DeleteRow">
   <xforms:trigger>
      ...xforms_label...
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the label control

The *label* control is a mandatory element of the *trigger* control. It provides text for the button. However, if you prefer use to use the XFDL value option to label the button, the content of *label* may be left blank. If the *label* control has a value, it overrides the XFDL *value* option.

The format of the *label* control is:

```
<xforms:label>button_text</xforms:label>
```

You can use *label* to set the text that the button displays, such as Delete Row. For example:

```
<xforms:label>Delete Row</xforms:label>
```

The *label* control is nested inside the *trigger* control. For example:

```
<xforms:trigger>
   <xforms:label>Delete Row</xforms:label>
</xforms:trigger>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="DeleteRow">
   <xforms:trigger>
      <xforms:label>Delete Row</xforms:label>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the delete action

The *delete* action deletes specified nodes from the instance data. If you delete a parent node, its children are deleted as well.

The format of the *delete* action is:

```
<xforms:delete ...attributes.../>
```

Delete has three attributes:

1. **at** — Determines the location of the items to be deleted. This attribute usually refers to an XFDL group-type item, such as *pane, table,* or *group*. The format of *at* is:
   ```
   at="index('item_sid')"
   ```
2. **nodeset** — Indicates the node or nodeset to be deleted. The format of *nodeset* is:
   ```
   nodeset="node"
   ```
3. **event** — Indicates the type of event that is triggered. In the case of *insert*, it is always:
   ```
   ev:event="DOMActivate"
   ```

An example of a *delete control* with all of its attributes is:

```
<xforms:delete
   at="index('customer_table')"
   nodeset="customer"
   ev:event="DOMActivate"/>
```

Delete is placed inside of the trigger control that activates the action. For example:

```
<xforms:trigger>
   <xforms:label>Delete Row</xforms:label>
   <xforms:delete
      at="index('customer_table')"
      nodeset="customer"
      ev:event="DOMActivate"/>
</xforms:trigger>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="DeleteRow">
   <xforms:trigger>
      <xforms:label>Delete Row</xforms:label>
      <xforms:delete
         at="index('customer_table')"
         nodeset="customer"
         ev:event="DOMActivate"/>
   </xforms:trigger>
   ...XFDL options...
</button>
```

Example of a completed delete row button

The following code shows a sample of a button that removes a row from a table:

```
<button sid="DeleteRow">
   <xforms:trigger>
      <xforms:label>Delete Row</xforms:label>
      <xforms:delete
         at="index('customer_table')"
         nodeset="customer"
         ev:event="DOMActivate"/>
   </xforms:trigger>
   <fontcolor>blue</fontcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Creating fields

Generally, you use fields to collect information from the user, such as names, dates, dollar amounts, and so on. You can set up fields to check and restrict users' entries, to flag errors and omissions and provide help on how to correct them, to format user input in a standard style, and to perform calculations and make logical decisions.

You can use form controls to create four different kinds of fields:

- Single line fields
- Multi-line fields
- Password fields

- Rich Text Fields

Note: To create a readonly field, use the readonly model item property. For detailed information about the readonly model item property, see "Readonly" on page 19.

Creating single line fields

Single line fields allow users to enter only a single line of text. This is done using an *input* control. *Input* limits a field's text area to a single line and links it to a node in the data instance.

To create a single line field, you must:

- Create the data node
- Create the XFDL field.
- Add the *input* control.
- Add the *label* control.

Create the data node

As fields simply collect user input, you need to create a node in the data instance to contain the user's data. You can name this node anything you want, although it should describe the purpose of the node. In the example for this section, we will create a field that requests the user's name. Therefore, we will name the node <name>. For example:

```
<name></name>
```

Create the XFDL field

The single line field is written as a normal XFDL *field* item. Using XFDL options, you can modify the field's appearance and position any way you want. This includes determining size of the field and the color of its text.

For example, you might create the following field:

```
<field sid="Name">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</field>
```

Add the input control

The *input* control limits a field's text area to a single line and links it to a node in the data instance. The format of the *input* control is:

```
<xforms:input attribute>...xforms_label...</xforms:input>
```

Input has three attributes:

1. **ref** — The path to the instance node. This attribute links the *input* control to the form node. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

For example, if the *input* control was associated with the <name> node inside the <personal_info> instance, then the XPath reference would be:

```
ref="instance('personal_info')/name"
```

2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was ″model2″, then you would write:

   ```
   model="model2"
   ```

3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the field. *Input* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

The input control is nested inside the XFDL field item, as shown:

```
<field sid="Name">
   <xforms:input ref="instance('personal_info')/name">
      ...xforms_label...
   </xforms:input>
   ...XFDL options...
</field>
```

Add the label control

The *label* control is a mandatory element of the *input* control. It is intended to provide text for the field’s label. However, if you prefer use to use an XFDL *label* item to label the field, the content of the *label* control may be left blank. If the *label* control has a value, it overrides the XFDL *label* item.

The format of the *label* control is:

```
<xforms:label>text</xforms:label>
```

The *label* control is nested inside the *input* control. For example:

```
<xforms:input>
   <xforms:label>Name</xforms:label>
</xforms:input>
```

Both are contained inside the XFDL field, as shown:

```
<field sid="Name">
   <xforms:input ref="instance('personal_info')/name">
      <xforms:label>Name</xforms:label>
   </xforms:input>
   ...XFDL options...
</field>
```

Example of a completed single line field

The following code shows a sample of a field that is limited to a single line:

```
<field sid="Name">
   <xforms:input ref="instance('personal_info')/name">
      <xforms:label>Name</xforms:label>
   </xforms:input>
   <fontcolor>blue</fontcolor>
   <itemlocation>
```

```
         <x>0</x>
         <y>23</y>
      </itemlocation>
      <size>
         <width>20</width>
         <height>10</height>
      </size>
      <value></value>
   </field>
```

Creating multi-line fields

Multi-line fields are not limited in the number of lines they display. This is done using the *textarea* control. *Textarea* also links the field to a node in the data instance.

To create a multi line field, you must:

- Create the data node.
- Create the XFDL field.
- Add the *textarea* control.
- Add the *label* control.

Create the data node

As fields simply collect user input, you need to create a node in the data instance to contain the user's data. You can name this node anything you want, although it should describe the purpose of the node. In the example for this section, we will create a field that collects additional comments. Therefore, we will name the node <comments>. For example:

```
   <comments></comments>
```

Create the XFDL field

The multi line field is written as a normal XFDL *field* item. Using XFDL options, you can modify the field's appearance and position any way you want. This includes determining size of the field and the color of its text.

For example, you might create the following field:

```
   <field sid="Comments">
      <itemlocation>
         <x>0</x>
         <y>23</y>
      </itemlocation>
      <size>
         <width>20</width>
         <height>10</height>
      </size>
      <value></value>
   </field>
```

Add the textarea control

The *textarea* control links the field to a node in the data instance. *Textarea* does not limit the size of the field.

The format of the *textarea* control is:

```
   <xforms:textarea attribute> ...xforms_label... </xforms:textarea>
```

Textarea has three attributes:

1. **ref** — The path to the instance node. This attribute links the *textarea* control to the form node. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *textarea* control was associated with the <comments> node inside the personal_info instance, then the XPath reference would be:

   ```
   ref="instance('personal_info')/comments"
   ```
2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was "model2", then you would write:

   ```
   model="model2"
   ```
3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the field. *Textarea* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was "setRelevant", then you would write:

   ```
   bind="setRelevant"
   ```

The textarea control is nested inside the XFDL field item:

```
<field sid="Comments">
   <xforms:textarea ref="instance('personal_info')/comments">
      ...xforms_label...
   </xforms:textarea>
   ...XFDL options...
</field>
```

Add the label control

The *label* control is a mandatory element of the *textarea* control. It is intended to provide text for the field's label. However, if you prefer use to use an XFDL *label* item to label the field, the content of the *label control* may be left blank. If the *label* control has a value, it overrides the XFDL *label* item.

The format of the *label* control is:

```
<xforms:label>text</xforms:label>
```

The *label* control is nested inside the *textarea* control. For example:

```
<xforms:textarea>
   <xforms:label>Comments</xforms:label>
</xforms:textarea>
```

Both are contained inside the XFDL field, as shown:

```
<field sid="Comments">
   <xforms:textarea ref="instance('personal_info')/comments">
      <xforms:label></xforms:label>
   </xforms:textarea>
   ...XFDL options...
</field>
```

Example of a completed multi-line field

The following code shows a sample of a multi line field:

```
<field sid="Comments">
   <xforms:textarea ref="instance('personal_info')/comments">
      <xforms:label>Comments</xforms:label>
   </xforms:textarea>
   <fontcolor>blue</fontcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</field>
```

Creating password fields

Password fields allow users to enter write-only data. Any text typed by the users appears in the field as a line of asterisks, which prevents others from seeing the user's password.

To create password field, you must use the *secret* control. The *secret* control ensures that the user's input is displayed as asterisks, limits the field's text area to a single line, and links the data to a node in the data instance.

To create a password field, you must:

- Create the data node.
- Create the XFDL field.
- Add the *secret* control.
- Add the *label* control.

Create the data node

As fields simply collect user input, you need to create a node in the data instance to contain the user's data. You can name this node anything you want, although it should describe the purpose of the node. In the example for this section, we will create a field that requests the user's password. Therefore, we will name the node <password>. For example:

```
<password></password>
```

Create the XFDL field

The password field is written as a normal XFDL *field* item. Using XFDL options, you can modify the field's size and positioning any way you want.

For example, you might create the following field:

```
<field sid="Password">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</field>
```

Add the secret control

The *secret* control ensures that any data a user enters into the field is represented by asterisks. The correct data is still passed to the data node, but protected from view within the form. *Secret* also limits the field to a single line.

The format of the *secret* control is:

```
<xforms:secret attribute>...xforms_label...</xforms:secret>
```

Secret has three attributes:

1. **ref** — The path to the instance node. This attribute links the *secret* control to the form node. The format of *ref* is:
   ```
   ref="path_to_node"
   ```
   For example, if the *secret* control was associated with the <password> node inside the <personal_info> instance, then the XPath reference would be:
   ```
   ref="instance('personal_info')/password"
   ```
2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:
   ```
   model="model_id"
   ```
   For example, if the model id was ″model2″, then you would write:
   ```
   model="model2"
   ```
3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the field. *Secret* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:
   ```
   bind="bind_id"
   ```
   For example, if the bind id was ″setRelevant″, then you would write:
   ```
   bind="setRelevant"
   ```

The *secret* control is nested inside the XFDL field item:

```
<field sid="Password">
   <xforms:secret ref="instance('personal_info')/password">
      ...xforms_label...
   </xforms:secret>
   ...XFDL options...
</field>
```

Add the label control

The *label* control is a mandatory element of the *secret* control. However, if you prefer use to use an XFDL *label* item to label the field, the content of the *label* control may be left blank. If the *label* control has a value, it overrides the XFDL *label* item.

The format of the *label* control is:

```
<xforms:label>text</xforms:label>
```

The *label* control is nested inside the *textarea* control. For example:

```
<xforms:secret>
   <xforms:label>Password</xforms:label>
</xforms:secret>
```

Both are contained inside the XFDL field, as shown:

```
<field sid="Password">
   <xforms:secret ref="instance('personal_info')/password">
      <xforms:label>Password</xforms:label>
   </xforms:secret>
   ...XFDL options...
</field>
```

Example of a completed password field

The following code shows a sample of a password field:

```
<field sid="Password">
   <xforms:secret ref="instance('personal_info')/password">
      <xforms:label>Password</xforms:label>
   </xforms:secret>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</field>
```

Creating labels

A label is an XFDL item that displays text or images and does not accept user input. Labels are normally used as titles or to identify or describe another item that does require user input, such as fields or radio buttons.

There are two kinds of XForms labels:

- Stand-alone text and image labels.
- Text labels that describe another item.

There are also XFDL *label* items (that do not contain XForms controls), which offer more flexibility for appearance and positioning than XForms *label* controls. You will probably find that there are many situations where you prefer to use XFDL *labels* instead of *label* controls. Best practices for use are discussed in the following sections.

Creating stand-alone labels

Stand-alone labels are frequently used as titles, to display the results of calculations, or to display images. They are not associated with any other item. These labels may be strictly XFDL *label* items or they may contain an *output* control. If a stand-alone label has static content, you may prefer to use only XFDL options to create it. However, if the label gets its content from a data node, it must contain an *output* control.

To create a stand-alone label with content from the data instance, you must:

- Create the data node.
- Create an XFDL label.
- Add the *output* control.

Create the data node

As labels simply display data, you only need to create a node in the data instance to contain the information you want to display. You can name this node anything you want, although it should describe the purpose of the node. In the example for this section, we will create a label that displays the Grand Total of a purchase form. Therefore, we will name the node <grand_total>. For example:

```
<grand_total></grand_total>
```

Create the XFDL label

The stand-alone label is written as a normal XFDL *label* item. Using XFDL options, you can modify the label's size, appearance, and positioning any way you want.

For example, you might create the following label:

```
<label sid="GrandTotal">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</label>
```

Add the output control

The *output* control provides an image or text for an XFDL *label* item through its link to a data node.

The format of the *output* control is:

```
<xforms:output attribute></xforms:output>
```

Output has three attributes:

1. **ref** — The path to the instance node. This attribute links the *output* control to the form node. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *output* control was associated with the <grand_total> node, then the XPath reference would be:

   ```
   ref="grand_total"
   ```

2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was ″model2″, then you would write:

   ```
   model="model2"
   ```

3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the label. *Output* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

The *output* control is nested inside the XFDL *label* item :

```
<label sid="GrandTotal">
   <xforms:output ref="grand_total">
   </xforms:output>
   ...XFDL options...
</label>
```

An example of a completed stand-alone label

The following code shows a sample of a stand-alone label:

```
<label sid="GrandTotal">
   <xforms:output ref="grand_total">
   </xforms:output>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</label>
```

Creating labels describing specific items

The majority of labels specifically describe other items, such as fields, radio buttons, and so on. In XForms, the need for such labeling is recognized by the *label* control. Most XForms controls, such as *input, secret,* and *textarea* require a mandatory *label* control as part of their code. However, the XForms *label* control does not allow for special positioning or appearance options. For this reason, many form designers prefer to leave the *label* control blank, and use an XFDL *label* item to label the item. However, if your label gets its content from the data instance, your must use an XForms control.

To create a *label* control that references a data node, you must:

- Create the data node (optional).
- Create the item you want to label.
- Add its corresponding form control.
- Add a *label* control.

Create the data node

As labels simply display data, you only need to create a node in the data instance to contain the information you want to display. You can name this node anything you want, although it should describe the purpose of the node. Note that if you plan to leave the xforms label blank and replace it with an XFDL label, you do not need to create a data node for the label.

In the example for this section, we want to create a label for the "Last Name" field. Therefore, we will name the node <lname_label>. For example:

```
<lname_label></lname_label>
```

Create the XFDL item you want to label

Every item that allows user input on a form needs to have a label. In this example, we are using a field, but labels can be used with any item.

An example of a basic XFDL field is:

```
<field sid="LastName">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</field>
```

Add the corresponding form control

Form controls link the XFDL item to the XForms data instance. Without the form control, there is no link and the item cannot receive data from nor send data to the instance. The type of form control you use depends on the type of item you want to create. For example, if you wanted to create a multi-line field, you would use *textarea*. If you wanted to create a list, you would use *select1*. For the purposes of our example, we are creating a single line field, so we will use *input*.

The *input* control limits a field's text area to a single line and links it to a node in the data instance. The format of the *input* control is:

```
<xforms:input attribute>...xforms_label...</xforms:input>
```

Input has three attributes:

1. **ref** — The path to the instance node. This attribute links the *input* control to the form node. The format of *ref* is:
   ```
   ref="path_to_node"
   ```
   For example, if the *input* control was associated with the <name> node inside the personal_info instance, then the XPath reference would be:
   ```
   ref="instance('personal_info')/lname_label"
   ```
2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:
   ```
   model="model_id"
   ```
   For example, if the model id was ″model2″, then you would write:
   ```
   model="model2"
   ```
3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the field. *Input* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:
   ```
   bind="bind_id"
   ```
   For example, if the bind id was ″setRelevant″, then you would write:
   ```
   bind="setRelevant"
   ```

The *input* control is nested inside the XFDL field item:

```
<field sid="LastName">
   <xforms:input ref="instance('personal_info')/lname_label">
      ...xforms_label...
   </xforms:input>
   ...XFDL options...
</field>
```

Add the label control

The *label* provides a label for items. It is equivalent to an XFDL *label* option, although form developers cannot specify a *label* controls appearance.

The format of the *label* control is:

```
<xforms:label ...XPath Ref...>text</xforms:label>
```

The *label* control is a mandatory element inside many other form controls, including *input*, *secret*, *select*, *select1*, *submit*, *textarea*, *trigger*, and *upload.*

Label has two attributes:

1. **ref** — The path to the instance node. This attribute links the *input* control to the form node. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *input* control was associated with the <lname_label> node inside the personal_info instance, then the XPath reference would be:

   ```
   ref="instance('personal_info')/lname_label"
   ```

2. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the label. If the label does not have a static value, then *xforms:label* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

Ref and *bind* are only used if you want the content of this label to be controlled by the data node. For example, if the content of the data node is supplied by another item, you would use the *ref* attribute to associate the label with that data. An example of a label with a *ref* attribute is:

```
<xforms:label ref="instance('personal_info')/lname_label"/>
```

If you want the label to have a static value, do not use a *ref* or *bind* attribute. For example:

```
<xforms:label>Last Name</xforms:label>
```

The *label* control is nested inside the form control it describes. In this case, *input*. For example:

```
<field sid="LastName">
   <xforms:input ref="instance('personal_info')/last_name">
      <xforms:label ref="instance('personal_info')/lname_label"/>
   </xforms:input>
   ...XFDL options...
</field>
```

An example of a completed item with a label control

The following code shows a sample of a field with a *label* control:

```
<field sid="LastName">
   <xforms:input ref="instance('personal_info')/last_name">
      <xforms:label ref="instance('personal_info')/lname_label">
      </xforms:label>
   </xforms:input>
   <itemlocation>
      <x>0</x>
```

```
         <y>23</y>
      </itemlocation>
      <size>
         <width>20</width>
         <height>10</height>
      </size>
      <value></value>
   </field
```

Creating lists

Lists provide you with various different ways to present multiple choices to users. Each one serves a different purpose. Lists include:

- **Check Boxes —** Use *check boxes* with the *select* control if you want the user to be able to select more than one of the choices you present.
- **Exclusive Check Boxes —** Use exclusive *check boxes* with the *select1* control if you want the appearance of a check list, but only allow the user to select one option.
- **Radio Buttons —** Use *radio buttons* with the *select1* control if you:
  - Want the user to select only one choice.
  - Want to show all of the possible choices at once.
  - Want the user to have to click just once to select the choice.
  - Have a limited number of choices.
- **Lists —** Use *lists* with the *select1* control if you:
  - Want the user to select only one choice.
  - Want to save space on your form.
  - Want to have your application dynamically generate lists of choices and insert them into the form.

You can choose from three kinds of lists: popup lists, combo box lists, and box lists.

- **Popup List** — A popup list appears as a single field on your form. It allows the user to select a single choice.
- **Combo box list** — A combo box list appears as a single field on your form. It allows the user to type in a choice or choose one from a popup list.
- **Box list** — A box list can be any size and displays choices in a scrolling list.

As well as providing multiple ways of displaying choices to users, XForms allows you to choose how you create lists. You have two options:

- **Auto-Generated List —** The list is generated directly from a data instance. There is an associated data node for each choice in the list.
- **List Items Specified in the User Interface —** In this type of list, you must create individual cells and labels inside the list items. This type of list is associated to a single data node that records the user's selection.

Creating an auto-generated list

When you create an auto-generated list, the content of the list is determined by the data model. To create an auto-generated list, you must create a new instance with data nodes that represent each choice in the list. These choices are linked with the XFDL list through the *itemset* control.

To create an auto-generated list:

- Create the instance that will contain the list selections.

- Create the XFDL list.
- Add the *select* or *select1* control.
- Add its *label* control.
- Add the *itemset* control.
- Add its *label* control.
- Add its *value* control.

Create the instance

The data instance contains the list selections as a group of nodes. The names of these nodes are arbitrary, but they must all be the same. Furthermore, the names should be representative of the purpose of the nodes, such as ″choice″ or ″option″. For the purposes of this example, we will use *choice*.

The format of our *choice* node is:

```
<choice attribute>database entry</choice>
```

The attribute of the *choice* node is arbitrary. Typically, it is best to choose an attribute name that describes the responsibility of the node. As the content of the attribute will be what is displayed in the list, the attribute name is frequently ″show″ or ″display″. For example:

```
<choice show="option">database entry</choice>
```

While you may use any attribute name you like, remember that this attribute must be later referenced by form controls in the list.

The content of the attribute depends on what choice you want to display on the screen. For example, if your form is collecting the user's donation amount, the choices would reflect possible contribution amounts, such as $15, $25, $50, and so on. For example:

```
<choice show="$25">database entry</choice>
<<<<<<< i_xforms_g_creating_lists_auto-generated.dita
```

```
=======
```

The text that goes between the two *choice* tags is the text that you want to have appear in data instance or in your database. This text may be the same as the option or an abbreviated code, such as `25'. For example:

```
>>>>>>> 1.2
<choice display="$25">25</choice>
```

The following example shows an instance that contains a list of donation options:

```
<xforms:instance xmlns="" id="donation">
   <options>
         <choice display="$15">15</choice>
         <choice display="$25">25</choice>
         <choice display="$50">50</choice>
         <choice display="$75">75</choice>
      <choice display="$100">100</choice>
      </options>
</xforms:instance>
```

Create the XFDL list

The XFDL wrapper for a list created from the data instance can be any XFDL list type, such as checkgroups, comboboxes, or box lists.

For example, you might create the following list:

```
<combobox sid="Donations">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <bgcolor>bisque</bgcolor>
   <fontcolor>119,106,91</fontcolor>
</combobox>
```

Add the select or select1 control

To create a list from a data instance, you must use the appropriate *select* or *select1* controls. The *select* control is used to create a group that allows users to select more than one option, while the *select1* control limits users to a single choice. For the purposes of our example, we are using *select1*.

The format of the *select1* control is:

```
<xforms:select1 attributes>
   ...xforms_itemset...
   ...xforms_label...
</xforms:select1>
```

Select1 has five attributes:

1. **appearance** — Determines how the list is displayed to the user. The format of *appearance* is always:

   ```
   appearance="setting"
   ```

   There are three possible settings for *appearance*:

   - **full** — Expands the list so that the entire list is always visible. Use with checkgroups and radiogroups.
   - **minimal** — Limits the list to one row in height unless it is being accessed by a user. Use with popups and comboboxes.
   - **compact** — Displays the list as a framed box list. Use with box lists only.
2. **ref** — The path to the instance node . This attribute links the *select1* control to the form node that will accept user input. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *select1* control was associated with the <donation> node in the <customer_info> instance, then the XPath reference would be

   ```
   ref="instance('customer_info')/donation"
   ```
3. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was ″model2″, then you would write:

   ```
   model="model2"
   ```
4. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the list. *Select* and *select1* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```
5. **selection** — Determines whether the user can add their own entry to the list. This attribute is used with comboboxes only. The format of *selection* is always:

```
selection="open"
```

An example of a completed *select1* control for a combobox is:

```
<xforms:select1 appearance="minimal"
   selection="open"
   ref="instance('customer_info')/donation">
   ...xforms_label...
   ...xforms_itemset...
</xforms:select1>
```

The select1 control is nested inside the XFDL combobox item:

```
<combobox sid="Donations>
   <xforms:select1 appearance="minimal"
   selection="open"
   ref="instance('customer_info')/donation">
      ...xforms_label...
      ...xforms_itemset...
   </xforms:select1>
   ...XFDL options...
</combobox>
```

Add the label control

The *label* control is a mandatory element of the *select* and *select1* controls. It is used to provide a label for the entire list. As you cannot control the appearance or placement of the *label* control, you may prefer to leave the *label* control blank and replace it with a customizable XFDL *label* item.

The format of the *label* control is:

```
<xforms:label>text</xforms:label>
```

The *label* control is nested inside the *select* and *select1* controls. For example:

```
<xforms:select1 appearance="minimal"
   selection="open"
   ref="instance('customer_info')/donation">
   <xforms:label>   Donation Amount   </xforms:label>
   ...xforms_itemset...
</xforms:select1>
```

Add the itemset control

The *itemset* control is used with list-type items, such as checkgroups, radiogroups, and comboboxes. In combination with other controls such as *select* and *select1*, *itemset* allows you to create lists and groups that offer dynamic choices that are determined while the form is in use.

The format of the *itemset* control is:

```
<xforms:itemset attribute>
   ...xforms:label...
   ...xforms:value...
</xforms:itemset>
```

Itemset has one attribute:

- **Nodeset —** The path to the group of nodes that provide the list options. The format of *nodeset* is:

  ```
  nodeset="path to nodes"
  ```

  For example, if the *itemset* control was associated with the <choice> nodes inside the donation instance, then the nodeset would be:

```
    nodeset="instance('donation')/choice"
```

An example of a completed *itemset* control is:

```
<xforms:itemset nodeset="instance('donation')/choice">
</xforms:itemset>
```

Itemset is nested inside the *select1* control. For example:

```
<xforms:select1 appearance="minimal"
   selection="open"
   ref="instance('customer_info')/donation">
   <xforms:label></xforms:label>
   <xforms:itemset nodeset="instance('donation')/choice">
   </xforms:itemset>
</xforms:select1>
```

Add the label control

The *label* control is a mandatory element of the *itemset* control. This label is used to provide the choices for the list. Using a reference to the attributes of the <choice> nodes, *label* creates and displays the list of choices that are displayed to the user.

The format of the *label* control is:

```
<xforms:label attribute/>
```

This *label* control must always contain a *ref* attribute that references the attributes of the data nodes that contain the list of choices. The format of *ref* is:

```
ref="@attribute"
```

For example, if the *label* control was associated with the *display* attribute of the <choice> nodes in the <donation> instance, then the XPath reference would be:

```
ref="@display"
```

An example of a complete *label* control for *itemset* is:

```
<xforms:label ref="@display"/>
```

The *label* control is nested inside the *itemset* control. For example:

```
<xforms:itemset nodeset="instance('donation')/choice">
   <xforms:label ref="@display"/>
   ...xforms:value...
</xforms:itemset>
```

Add the value control

The *value* control indicates which options are selected. It also collects user selections and passes them to the XFDL *value* option. The *select1* control collects this data from the XFDL *value* option and passes it to the data node.

The format of *value* is:

```
<xforms:value attribute></xforms:value>
```

Value has one attribute:

- **ref —** The path to the instance node . This attribute links the *select1* control to the form node that will accept user input. The *ref* attribute of a *value* control in a list is always:

  ```
  ref="."
  ```

An example of a completed *value* control is:

```
<xforms:value ref="."/>
```

Value is nested inside the *itemset* control. For example:

```
<xforms:itemset nodeset="instance('donation')/choice">
   <xforms:label ref="@display"/>
   <xforms:value ref="."/>
</xforms:itemset>
```

Example of a completed auto-generated list

The following example shows a completed combobox:

```
<combobox sid="Donations">
   <xforms:select1 appearance="minimal"
      selection="open"
      ref="instance('customer_info')/donation">
         <xforms:label></xforms:label>
      <xforms:itemset nodeset="instance('donation')/choice">
         <xforms:label ref="@display"/>
         <xforms:value ref="."/>
      </xforms:itemset>
   </xforms:select1>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <bgcolor>bisque</bgcolor>
   <fontcolor>119,106,91</fontcolor>
</combobox>
```

Creating a list with choices specified in the user interface

Specifying your list choices inside the UI allows you to minimize the number of nodes in your data instance. Instead of creating data nodes, you must add an *item* control to the list for every choice you want to offer.

To create a list using the *item* control:

- Create the data node.
- Create an XFDL list.
- Add the *select* or *select1* control.
- Add its *label* control.
- Add the first choice:
  - Add the *item* control.
  - Add its *label* control.
  - Add its *value* control.
- Add remaining choices.

Create the data node

As you are specifying your list choices in the user interface, you only need to create a node in the data instance to collect the user's selection. You can name these nodes anything you want, although it should describe the purpose of the node. In the example for this section, we will create a popup that offers menu selections. Therefore, we will name the node <menu_prefs>. For example:

```
<menu_prefs></menu_prefs>
```

Create the XFDL list item

The XFDL wrapper for a list with choices specified in the user interface can be any XFDL list type, such as checkgroups, comboboxes, or box lists.

For example, you might create the following list:

```
<popup sid="MenuPrefs>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <bgcolor>bisque</bgcolor>
   <fontcolor>119,106,91</fontcolor>
</popup>
```

Add the select or select1 control

To create a list-type item, you must use the appropriate *select* or *select1* control. The *select* control is used to create a group that allows users to select more than one option, while the *select1* control limits users to a single choice. For the purposes of our example, we are using *select1*.

The format of the *select1* control is:

```
<xforms:select1 attributes>
   ...xforms_items...
   ...xforms_label...
</xforms:select1>
```

Select1 has three attributes:

- **ref** — The path to the instance node . This attribute links the *select* control to the form node that will accept user input. The format of *ref* is:

  ```
  ref="path_to_node"
  ```

  For example, if the *select* control was associated with the <menu_prefs> node in the customer_info instance, then the XPath reference would be

  ```
  ref="instance('customer_info')/menu_prefs"
  ```

- **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

  ```
  model="model_id"
  ```

  For example, if the model id was "model2", then you would write:

  ```
  model="model2"
  ```

- **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the list. *Select* and *select1* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

  ```
  bind="bind_id"
  ```

  For example, if the bind id was "setRelevant", then you would write:

  ```
  bind="setRelevant"
  ```

- **appearance** — Determines how the *select1* control is displayed to the user. For popup, the format of *appearance* is always:

  ```
  appearance="minimal"
  ```

  However, there are three possible settings for *appearance*:

  - **full** — Expands the list so that the entire list is always visible. Use with checkgroups and radiogroups.

  - **minimal** — Limits the list to one row in height unless it is being accessed by a user. Use with popups and comboboxes.
  - **compact** — Displays the list as a framed box list. Use with box lists only.
- **selection** — Determines whether the user can add their own entry to the list. This attribute is used with comboboxes only. The format of *selection* is always:

```
selection="open"
```

An example of a completed *select1* control is:

```
<xforms:select1 appearance="minimal"
   ref="instance('customer_info')/menu_prefs">
   ...xforms_label...
   ...xforms_itemset...
</xforms:select1>
```

The select1 control is nested inside the XFDL popup item:

```
<popup sid="MenuPrefs>
   <xforms:select1 appearance="full"
      ref="instance('customer_info')/menu_prefs">
      ...xforms_label...
      ...xforms_items...
   </xforms:select1>
   ...XFDL options...
</popup>
```

Add the label control

The *label* control is a mandatory element of the *select* control. It is used to provide a label for the entire checkgroup. As you cannot control the appearance or placement of the *label* control, you may prefer to leave the *label* control blank and replace it with a customizable XFDL *label* item.

The format of the *label* control is:

```
<xforms:label></xforms:label>
```

The *label* control is nested inside the *select1* control. For example:

```
<xforms:select1 appearance="full" ref="menu_prefs">
   <xforms:label>      </xforms:label>
   ...xforms_items...
</xforms:select>
```

Add the item control for the first choice

The *item* control is used to create choices inside list-type items, such as checkgroups, radiogroups, and comboboxes.

The format of the *item* control is:

```
<xforms:item>
   ...xforms:label...
   ...xforms:value...
</xforms:item>
```

Item is nested inside the *select1* control. For example:

```
<xforms:select1 appearance="minimal"
   ref="instance('customer_info')/menu_prefs">
   <xforms:label>      </xforms:label>
   <xforms:item>
```

```
      ...xforms:label...
      ...xforms:value...
   </xforms:item>
</xforms:select1>
```

Add the label control for the first choice

The *label* control is used to provide the content of the popup. You place the name of the popup choice between the *label* tags.

The format of the *label* control is:

```
<xforms:label>choice</xforms:label>
```

For example, if you wanted your popup list to display a list of menu options, your label might read:

```
<xforms:label>Chicken</xforms:label>
```

The *label* control is nested inside the *item* control. For example:

```
<xforms:item>
   <xforms:label>Chicken</xforms:label>
   ...xforms:value...
</xforms:item>
```

Add the value control for the first choice

The *value* control indicates which data is passed to the instance if this choice is selected.

The format of *value* is:

```
<xforms:value>choice_data</xforms:value>
```

For example, if your menu database has three letter codes to represent each menu choice, then the *value* might be:

```
<xforms:value>chk</xforms:value>
```

Value is nested inside the *item* control. For example:

```
<xforms:item>
   <xforms:label>Chicken</xforms:label>
   <xforms:>chk</xforms:value>
</xforms:item>
```

Add remaining choices

You must create an *item* control, with nested *label* and *value* controls for each choice you want to have in your list. For example, if you wanted your menu to offer roast beef, chicken, salmon, or vegetarian options, your popup would contain the following four *item* controls with their corresponding *label* and *value* controls:

```
<xforms:item>
   <xforms:label>Roast Beef</xforms:label>
   <xforms:value>Beef</xforms:value>
</xforms:item>
<xforms:item>
   <xforms:label>Chicken</xforms:label>
   <xforms:value>chicken</xforms:value>
</xforms:item>
<xforms:item>
   <xforms:label>Salmon</xforms:label>
   <xforms:value>salmon</xforms:value>
   </xforms:item>
   <xforms:item>
      <xforms:label>Vegetarian</xforms:label>
      <xforms:value>veggie</xforms:value>
   </xforms:item>
```

Example of a completed popup

The following example shows a completed popup:

```
   <popup sid="MenuPrefs">
      <xforms:select1 appearance="minimal"        ref="menu_prefs">
            <xforms:label></xforms:label>
         <xforms:item>
            <xforms:label>Roast Beef</xforms:label>
            <xforms:value>Beef</xforms:value>
         </xforms:item>
         <xforms:item>
            <xforms:label>Chicken</xforms:label>
            <xforms:value>chicken</xforms:value>
         </xforms:item>
         <xforms:item>
            <xforms:label>Salmon</xforms:label>
            <xforms:value>salmon</xforms:value>
         </xforms:item>
         <xforms:item>
            <xforms:label>Vegetarian</xforms:label>
            <xforms:value>veggie</xforms:value>
         </xforms:item>
      </xforms:select1>
      <itemlocation>
         <x>0</x>
         <y>23</y>
      </itemlocation>
      <bgcolor>bisque</bgcolor>
      <fontcolor>119,106,91</fontcolor>
   </popup>
```

Creating date pickers

Date pickers are comboboxes that open to display a calendar rather than a list. The user can type a date into the combo box, or use the calendar to select a date. In either case, the date is displayed in the format of your choice.

To create a date picker:

- Create the data node.
- Create an XFDL combobox.
- Add an *input* control.
- Add its *label* control.

Create the data node

As date pickers simply collect user input, you need to create a node in the data instance to contain the date. You can name this node anything you want, although it should describe the purpose of the node. In the example for this section, we will create a combobox that requests an employee's hiring date. Therefore, we will name the node <hire_date>. For example:

```
   <hire_date></hire_date>
```

Create the XFDL combobox

Comboboxes display a list of choices and allow users to type in a choice if they want. When selecting from the list, users may only select one option. You should use comboboxes whenever you want to present a list of options, but also want to allow users to be able to enter another value as well.

For example, you might create the following combobox:

```
<combobox sid="HireDate">
   <format>
      <datatype>date</datatype>
   </format>
   <fontcolor>blue</fontcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</combobox>
```

Add the input control

The *input* control causes the combobox to open into a calendar instead of a list. It also limits the combobox's text area to a single line and links it to a node in the data instance.

The format of the *input* control is:

```
<xforms:input attribute> ...xforms_label... </xforms:input>
```

Input has three attributes:

1. **ref** — The path to the instance node. This attribute links the *input* control to the form node. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *input* control was associated with the <hire_date> node inside the personal_info instance, then the XPath reference would be:

   ```
   ref="instance('personal_info')/hire_date"
   ```

2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was ″model2″, then you would write:

   ```
   model="model2"
   ```

3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the combobox. *Input* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

The input control is nested inside the XFDL field item:

```
<combobox sid="HireDate">
   <xforms:input ref="instance('personal_info')/hire_date"
      ...xforms_label...
   </xforms:input>
   ...XFDL options...
</combobox>
```

Add the label control

The *label* control is a mandatory element of the *input* control. It is used to provide a label for the field. However, if you prefer use to use an XFDL *label* item to label the field, the content of the *label* control may be left blank. If the *label* control has a value, it overrides the XFDL *label* item.

The format of the *label* control is:

```
<xforms:label></xforms:label>
```

The *label* control is nested inside the *input* control. For example:

```
<xforms:input>
   <xforms:label></xforms:label>
</xforms:input>
```

Both are contained inside the combobox, as shown:

```
<combobox sid="HireDate">
   <xforms:input ref="instance('personal_info')/hire_date"
      <xforms:label></xforms:label>
   </xforms:input>
   ...XFDL options...
</combobox>
```

Example of a completed date picker

The following code shows a sample of a date picker:

```
<combobox sid="HireDate">
<xforms:input ref="instance('personal_info')/hire_date"
     <xforms:label></xforms:label>
</xforms:input>
   <format>
      <datatype>date</datatype>
   </format>
   <fontcolor>blue</fontcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</combobox>
```

Creating conditional items

Conditional items are items that are not displayed unless the proper conditions are met. For example, consider a form that needs to be rendered in several languages. You could have three labels (English, French, and Spanish) for each field, or you could create the labels and fields as conditional items and allow the user to select their language of preference.

Conditional items are always inside an XFDL *pane* item. Furthermore, each group of conditional items requires a Toggle Case button to activate them.

Steps to creating conditional items

The XFDL *pane* item provides a wrapper for two XForms form controls: *switch* and *case*. The *switch* control specifies what information is conditional. It also groups a set of *case* controls. The case controls contain the different data options. For example, if you want to have your form display in different languages, you can create a *case* for each language, allowing only the text for the selected language to be displayed.

To create conditional items:

- Create the associated nodes.
- Create an XFDL pane.
- Add the *switch* control.
- Add the *case* controls.
- Add the XFDL items to display in each case.
- Create a Toggle Case button for each case.

Create the associated nodes

When you create a *switch* control that contains conditional items, you must associate it with a master or *parent* data node. Essentially, associating a parent node with a *switch* control indicates that its child nodes have conditional associations. In other words, these nodes may be referenced by multiple conditional items in the user interface, but only retain the data provided by the active condition. As usual, the name of this node is entirely arbitrary, but it should be unique and reflect the purpose to which it will be used. In this example, our conditional items provide text in different languages for fields that collect the user's name and address. The *switch* control and its associated node group these nodes together. Therefore, we will call this node <switch_language>.

The format of this node is:

```
<switch_language>
   ...child_nodes...
</switch_language>
```

You must also create the child nodes. You must make one item node for each item that will appear in the pane. For example, if your conditional item pane will contain name, street, city, state, and ZIP code information, the <switch_language> node will need to have 5 children nodes; one for each piece of data:

```
<switch_language>
   <Name></Name>
      <Street></Street>
      <City></City>
      <State></State>
   <ZIP></ZIP>
</switch_language>
```

Create the XFDL pane

The pane provides a wrapper for the form controls that make items conditional.

An example of a typical XFDL pane is:

```
<pane sid="LanguageSwitch">
   <fontcolor>white</fontcolor>
   <bgcolor>cadet blue</bgcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>200</width>
      <height>100</height>
   </size>
</pane>
```

Add the switch control

The *switch* control is used to group one or more *case* controls. *Switch* also provides the XPath reference to the parent node of the associated *case* controls.

The format of the *switch* control is:

```
<xforms:switch attribute></xforms:switch>
```

The *switch* control always contains one or more *case* controls, as shown:

```
<xforms:switch attribute>
   ...xforms:case...
   ...xforms:case...
   ...xforms:case...
</xforms:switch>
```

The *switch* control has three attributes:

1. **ref** — Sets the new default starting node for every *ref* attribute contained in the *switch* control. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *switch* control was associated with the <switch_language> node inside the first data instance, then the XPath reference would be:

   ```
   ref="switch_language"
   ```

2. **model** — The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was ″model2″, then you would write:

   ```
   model="model2"
   ```

3. **bind** — The id of a bind whose nodeset is used to reference the data node you want to associate with the list. *Select* and *select1* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

The *switch* control is nested inside the XFDL pane, as shown:

```
<pane sid="LanguageSwitch">
   <xforms:switch ref="switch_language">
      ...xforms:case...
      ...xforms:case...
      ...xforms:case...
   </xforms:switch>
</pane>
```

Add the case controls

The *case* control encloses conditional form items. Essentially, it ensures that its contents are not displayed unless the proper conditions are met. For example, consider a form that needs to be rendered in several languages. You could have three labels (English, French, and Spanish) for each field, or you could create a *case* control for each language to ensure that only the user's preferred language is displayed. You need to create a different *case* for each set of conditional items.

The format of the *case* control is:

```
<xforms:case attribute>
   ...XFDL items...
</xforms:case>
```

Case has two attributes:

1. **id** — The name of the case. This name should be unique within the form and clearly indicate the purpose of the conditional items. The format of *id* is:

   ```
   ref="unique_name"
   <<<<<<< i_xforms_g_creating_conditional_items.dita
   ```

   For example, if the items defined by a *case* provide the Spanish version of a form, the control's name would be 'Spanish':

   ```
   =======
   ```

   For example, if the items defined by a *case* provide the Spanish version of a form, the control's name would be `Spanish':

   ```
   >>>>>>> 1.2
      id="Spanish">
   ```

2. **selection** — Determines the starting state of the *case* control. This attribute allows form designers to set a default case for the form. The format of *selection* is:

   ```
   selection="state"
   ```

   Selection has two possible settings: **true** and **false**. This attribute allows the form designer to set a default case for the form.

An example of a completed *case control* would be:

```
<xforms:case id="Spanish" selection="true">
   ...XFDL items...
</xforms:case>
```

Case controls are nested inside the *switch* controls. Note that you need a new *case* for each set of conditional items. For example:

```
<xforms:switch ref="switch_language">
   <xforms:case id="Spanish" selection="true">
      ...XFDL items...
   </xforms:case>
   <xforms:case id="English" selection="true">
      ...XFDL items...
   </xforms:case>
   <xforms:case id="French" selection="true">
      ...XFDL items...
   </xforms:case>
</xforms:switch>
```

Add the XFDL items for each case

Each *case* control contains the XFDL items that will be displayed when the case is selected. These can be any XFDL items, including fields, lists, and tables. The following example shows two fields from a Spanish case:

```
<field sid="Name_Spanish">
   <xforms:input ref="name">
      <xforms:label>Nombre del contribuidor</xforms:label>
   </xforms:input>
   ...XFDL options...
</field>
<field sid="Address_Spanish">
   <xforms:input ref="street">
      <xforms:label>Calle</xforms:label>
   </xforms:input>
   ...XFDL options...
</field>
```

Note: For explicit instructions for creating single line fields, see ″Creating Single Line Fields″ on page page Creating Single Line Fields.

Creating a toggle case button

The Toggle Case button allows users to alternate between conditional options, called *cases*. Essentially, Toggle Case buttons allow your users to select which case they want to use.

To create a Toggle Case button, you must:

- Create an XFDL button.
- Add the *trigger* control.
- Add the *label* control.
- Add the *toggle* action.

Create the XFDL button

The toggle case button's XFDL wrapper is a basic *button* item. Using XFDL options, you can modify the button's appearance and position any way you want. This includes determining size of the button and the appearance of its text.

An example of a basic XFDL button is:

```
<button sid="SwitchToSpanish">
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>20</width>
      <height>10</height>
   </size>
   <value></value>
</button>
```

Add the trigger control

The *trigger* control allows the user to trigger actions in the form. In this case, *trigger* is used to activate the *toggle* action. *Toggle* selects one possible *case* control from the possible cases in a *switch* group.

The format of the *trigger* control is:

```
<xforms:trigger></xforms:trigger>
```

The *trigger* control is nested inside the Toggle button item:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
      ...xforms:label...
     ...xforms:toggle...
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the label control

The *label* control is a mandatory element of the *trigger* control. It is intended to provide the text for the button.

The format of the *label* control is:

```
<xforms:label>button_text</xforms:label>
```

You can use *label* to set the text that the button displays, such as Switch To Spanish. For example:

```
<xforms:label>Switch To Spanish</xforms:label>
```

The *label* control is nested inside the *trigger* control. For example:

```
<xforms:trigger>
   <xforms:label>Switch To Spanish</xforms:label>
</xforms:trigger>
```

Both are contained inside the XFDL button, as shown:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
      <xforms:label>Switch To Spanish</xforms:label>
      ...xforms:toggle...
   </xforms:trigger>
   ...XFDL options...
</button>
```

Add the toggle action

The *toggle* action selects one possible case from all the possible cases contained in a *switch* group. In other words, *toggle* ensures that the selected case is turned on, while the other cases are turned off.

The format of *toggle* is:

```
<xforms:toggle attributes></xforms:toggle>
```

Toggle has two attributes:

1. **event** — Indicates the type of event that is triggered. This attribute is mandatory and is written as:

   ```
   ev:event="event"
   ```

   In the case of *toggle*, the event is usually **DOMActivate**.
2. **case** — Indicates which case is selected when the Toggle button is pushed. The format of *case* is:

   ```
   case="name"
   ```

This attribute must match the *id* of the *case* control you want to select. For example, if you wanted a case with an *id* of "Spanish" to be selected, you would write:

```
case="Spanish"
```

An example of a complete *toggle* action would be:

```
<xforms:toggle ev:event="DOMActivate" case="Spanish"></xforms:toggle>
```

The *toggle* action is nested inside the *trigger* control. For example:

```
<button sid="SwitchToSpanish">
      <xforms:trigger>
         <xforms:toggle ev:event="DOMActivate" case="Spanish"/>
         <xforms:label>Switch To Spanish</xforms:label>
      </xforms:trigger>
      ...XFDL options...
</button>
```

Example of a completed pane

The following code shows a sample of a pane containing conditional items:

```
<pane sid="LanguageSwitch">
   <xforms:switch ref="switch_language">
      <xforms:case id="English" selection="true">
         <field sid="Name_English">
            <xforms:input ref="name">
               <xforms:label>Name of contributor</xforms:label>
            </xforms:input>
            <itemlocation>
               <x>0</x>
               <y>23</y>
            </itemlocation>
         </field>
         <field sid="Address_English">
            <xforms:input ref="street">
               <xforms:label>Street Address</xforms:label>
            </xforms:input>
            <itemlocation>
               <after>Name_English</after>
            </itemlocation>
         </field>
         ...other address fields...
      </xforms:case>
      <xforms:case id="French" selection="false">
         <field sid="Name_French">
            <xforms:input ref="name">
               <xforms:label>Nom du contributeur</xforms:label>
            </xforms:input>
            <itemlocation>
               <x>0</x>
               <y>23</y>
            </itemlocation>
         </field>
         <field sid="Address_French">
            <xforms:input ref="street">
               <xforms:label>Rue</xforms:label>
            </xforms:input>
            <itemlocation>
               <after>Name_French</after>
            </itemlocation>
         </field>
         ...other address fields...
      </xforms:case>
      <xforms:case id="Spanish" selection="false">
```

```
         <field sid="Name_Spanish">
            <xforms:input ref="name">
               <xforms:label>Nombre del contribuidor</xforms:label>
            </xforms:input>
            <itemlocation>
               <x>0</x>
               <y>23</y>
            </itemlocation>
         </field>
         <field sid="Address_Spanish">
            <xforms:input ref="street">
               <xforms:label>Calle</xforms:label>
            </xforms:input>
            <itemlocation>
               <after>Name_Spanish</after>
            </itemlocation>
         </field>
         ...other address fields...
      </xforms:case>
   </xforms:switch>
</pane>
```

Example of completed toggle case buttons

The following code shows samples of completed toggle case buttons, one for each case in the pane above:

```
<button sid="SwitchToFrench">
   <xforms:trigger>
      <xforms:label>Français</xforms:label>
      <xforms:toggle case="French" ev:event="DOMActivate"/>
   </xforms:trigger>
   <itemlocation>
      <x>0</x>
      <y>50</y>
   </itemlocation>
</button>
<button sid="SwitchtoEnglish">
   <xforms:trigger>
      <xforms:label>English</xforms:label>
      <xforms:toggle case="English" ev:event="DOMActivate"/>
   </xforms:trigger>
   <itemlocation>
      <after>SwitchToFrench</after>
   </itemlocation>
</button>
<button sid="SwitchtoSpanish">
   <xforms:trigger>
      <xforms:label>Español</xforms:label>
      <xforms:toggle case="Spanish" ev:event="DOMActivate"/>
   </xforms:trigger>
   <itemlocation>
      <after>SwitchToEnglish</after>
   </itemlocation>
</button>
```

Creating groups

You can use the *pane* item to create groups of related items. Typically, you would group items if you wanted them all to be affected by the same computes, binds, properties, or other modifiers. For example, you may want to have a group of items that are all mutually *relevant* or change the background color of a pane depending upon whether any of its items have the focus.

The *pane* item is the XFDL wrapper for the XForms *group* control.

To create a group:

- Create the associated nodes.
- Create a pane.
- Add the *group* control.
- Add the XFDL items that you want in the group.

Create the associated nodes

When you create a group, you must associate it with a master or *parent* data node. Essentially, associating a parent node with a *group* control indicates that its child nodes are all part of the same group. As usual, the name of this node is entirely arbitrary, but it should be unique and reflect the purpose to which it will be used. In this example, our pane collects information that will only be *relevant* if the user indicates that they have a spouse or children. Therefore, we will call the group node <dependants>.

The format of this node is:

```
<dependants>
   ...child_nodes...
</dependants>
```

You must also create the child nodes. You must make one item node for each item that you want in the group. For example, if your group will contain the name of users' spouse and children, the <dependants> node will need to have a node for a spouse and each of their children; one for each piece of data. (Note that in this case, you may also want to give your users the ability to add or remove dependants.) For example:

```
<dependants>
   <spouse></spouse>
      <child1></child1>
      <child2></child2>
</dependants>
```

Create the XFDL pane

The pane provides an XFDL wrapper for the *group* control.

An example of a typical XFDL pane is:

```
<pane sid="Dependants">
   <fontcolor>white</fontcolor>
   <bgcolor>cadet blue</bgcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>200</width>
      <height>100</height>
   </size>
</pane>
```

Add the group control

The *group* control groups XFDL items inside a *pane* item and links them with a data node. Grouping items with this control ensures that they are all affected by the computes, binds, properties, or other modifiers that you want to have affect the group.

The format of the *group* control is:

```
<xforms:group attribute>XFDL items</xforms:group>
```

Group has three attributes:

1. **ref —** The path to the group node that groups the item nodes. *Group* must have either a *ref* attribute or a *bind* attribute. The format of *ref* is:

   ```
   ref="path_to_node"
   ```

   For example, if the *group* control was associated with the <dependants> node inside the primary instance, then the reference would be:

   ```
   ref="dependants"
   ```

2. **model —** The id of the XForms model that contains the instance you want to reference. This attribute is optional. If *model* is not included, it will default to the first model. The format of model is:

   ```
   model="model_id"
   ```

   For example, if the model id was ″model2″, then you would write:

   ```
   model="model2"
   ```

3. **bind —** The id of a bind whose nodeset is used to reference the data node you want to associate with the group. *Group* must have either a *bind* attribute or a *ref* attribute. The format of *bind* is:

   ```
   bind="bind_id"
   ```

   For example, if the bind id was ″setRelevant″, then you would write:

   ```
   bind="setRelevant"
   ```

An example of a completed *group* control could be:

```
<xforms:group ref="dependants">
   ...XFDL items...
</xforms:group>
```

The *group control* is nested inside a pane, as shown:

```
<pane sid="Dependants">
   <xforms:group ref="dependants">
      ...XFDL items...
   </xforms:group>
</pane>
```

Add the XFDL items

Every *group* control contains a number of XFDL items. These items can be any XFDL items, such as fields, comboboxes, or buttons. For the purposes of this example, we will create fields for user's to enter the names of their spouse and children:

```
<field sid="spouse_name">
   <xforms:input ref="spouse">
      <xforms:label>Spouse's Name</xforms:label>
   </xforms:input>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
</field>
<field sid="child1_name">
   <xforms:input ref="child1">
      <xforms:label>Child's Name</xforms:label>
   </xforms:input>
   <itemlocation>
      <x>0</x>
```

```
         <y>23</y>
      </itemlocation>
   </field>
   <field sid="child2_name">
      <xforms:input ref="child2">
         <xforms:label>Child's Name</xforms:label>
      </xforms:input>
      <itemlocation>
         <x>0</x>
         <y>23</y>
      </itemlocation>
   </field>
```

The XFDL items are nested inside the *group* control, as shown:

```
   <xforms:group ref="dependants">
      <field sid="spouse_name">
         <xforms:input ref="spouse">
            <xforms:label>Spouse's Name</xforms:label>
         </xforms:input>
         <itemlocation>
            <x>0</x>
            <y>23</y>
         </itemlocation>
      </field>
      ...other fields...
   </xforms:group>
```

Note: For detailed information about creating specific XFDL items, see the specific sections that describes how to create them, such as ″Creating Single Line Fields″ or″Creating an Auto-Generated List″ .

Example of completed group

The following code shows a sample of a completed group:

```
   <pane sid="Dependants">
      <xforms:group ref="dependants">
            <field sid="spouse_name">
               <xforms:input ref="spouse">
                  <xforms:label>Spouse's Name</xforms:label>
               </xforms:input>
               <itemlocation>
                  <x>0</x>
                  <y>23</y>
               </itemlocation>
            </field>
            <field sid="child1_name">
               <xforms:input ref="child1">
                  <xforms:label>Child's Name</xforms:label>
            </xforms:input>
            <itemlocation>
               <x>0</x>
               <y>33</y>
            </itemlocation>
         </field>
         <field sid="child2_name">
            <xforms:input ref="child2">
               <xforms:label>Child's Name</xforms:label>
            </xforms:input>
            <itemlocation>
               <x>10</x>
               <y>33</y>
            </itemlocation>
         </field>
      </xforms:group>
   <fontcolor>white</fontcolor>
   <bgcolor>cadet blue</bgcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>200</width>
      <height>100</height>
   </size>
</pane>
```

Creating tables

Tables allow you to arrange data into rows of items. This is done by creating a template row that includes all of the items that should appear in each row. You can place any item, such as fields, popups, and images inside a table, including other tables. Grouping items in a table can make data easier to interpret and forms easier to complete.

To create a table:

- Create the associated nodes in the data model.
- Create an XFDL table.
- Add the *repeat* control.
- Add the XFDL items that you want in the table.

Create the associated nodes

When you create a table, it must have data nodes associated with every row and item in the table. The names of these nodes are arbitrary, but the names of the row nodes must all be the same. Furthermore, all of the names should be representative of the purpose of the nodes to help you keep track of them. In this example, we will make a table of names and addresses, with one row for each person's information. Thus, we will call the row node *person*.

The format of this node is:

```
<person>
   ...item nodes...
</person>
```

You must make one item node for each item in your table row. For example, if your row will contain name, street, city, state, and ZIP code information, the *person* node will need to have 5 children nodes; one for each piece of data:

```
<person>
   <name></name>
      <street></street>
      <city></city>
      <state></state>
   <zip></zip>
</person>
```

If you want your table to open with multiple rows, you must add more groups of row nodes. The following example shows the nodes for a table with two rows:

```
<person>
   <name></name>
      <street></street>
      <city></city>
      <state></state>
   <zip></zip>
```

```
</person>
<person>
   <name></name>
      <street></street>
      <city></city>
      <state></state>
   <zip></zip>
</person>
```

Note: You do not have to create row nodes for rows that you don't exist when the form is opened. Those will be automatically generated when the user clicks the Insert Row button. For detailed information on creating an Insert Row button, see"Creating an insert button" on page 29.

Create the XFDL table

The *table* item provides an XFDL wrapper for the *repeat* control. Its options determine the appearance and location of the table, as well as allowing you to set which item receives the focus when users first tab into the form.

An example of a typical table is:

```
<table sid="Table1">
   <first>F2</first>
   <fontcolor>white</fontcolor>
   <bgcolor>cadet blue</bgcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <border>on</border>
</table>
```

Add the repeat control

The *repeat* control associates elements inside a table with nodes in a data instance.

The format of the *repeat* control is:

```
<xforms:repeat attribute></xforms:repeat>
```

Repeat has the following attributes:

- **id** — The name of the table. This optional attribute should be a unique name that describes the function of the table. The format of ***id*** is:

  ```
  id="name"
  ```

- **startindex** — Sets which row of the table gets the focus when the form is opened. This attribute is optional. If ***startindex*** is not used, the focus defaults to the first row in the table. The format of ***startindex*** is:

  ```
  startindex="number"
  ```

- **nodeset** — Indicates the data nodes associated with the *repeat* control. *Repeat* binds to these nodes and generates the table items from them when the form is opened. The format of *nodeset* is:

  ```
  nodeset="nodes"
  ```

  For example, if the *repeat* control was associated with the parent node <person> (with child nodes of <name>, <street>, <city> and so on) inside the personal_info instance, then the nodeset would be:

  ```
  nodeset="person"
  ```

If the *repeat* control was associated with the <person> nodeset (with attributes of name, street, city, and so on) inside the personal_info instance, then the XPath reference would also be:

```
nodeset="person"
```

An example of a completed *repeat* control would be:

```
<xforms:repeat id="Address_Table"
   startindex="2"
   nodeset="person">
   ...XFDL items...
</xforms:repeat>
```

The *repeat* control is nested inside the table item, as shown:

```
<table sid="T1">
   <xforms:repeat id="Address_Table"
   nodeset="person">
      ...XFDL items...
   </xforms:repeat>
   ...XFDL options...
</table>
```

Add the XFDL items

Every *group* control contains a number of XFDL items. These items can be any XFDL items, such as fields, comboboxes, or buttons. For the purposes of this example, the following sample shows several fields :

```
<field sid="Name">
   <xforms:input ref="name">
      <xforms:label>Name</xforms:label>
</field>
<field sid="Street">
   <xforms:input ref="street">
      <xforms:label>Street</xforms:label>
</field>
<field sid="City">
   <xforms:input ref="city">
      <xforms:label>City</xforms:label>
</field>
<field sid="State">
   <xforms:input ref="state">
      <xforms:label>State</xforms:label>
</field>
<field sid="ZIP">
   <xforms:input ref="zip">
      <xforms:label>Zip Code</xforms:label>
</field>
```

These XFDL items are nested inside the *repeat* control, as shown:

```
<xforms:repeat ref="person">
   <field sid="Name">
      <xforms:input ref="Name">
         <xforms:label>Name</xforms:label>
   </field>
   <field sid="Street">
      <xforms:input ref="Street">
         <xforms:label>Street</xforms:label>
   </field>
   <field sid="City">
      <xforms:input ref="City">
         <xforms:label>City</xforms:label>
   </field>
   <field sid="State">
```

```
         <xforms:input ref="State">
            <xforms:label>State</xforms:label>
      </field>
      <field sid="ZIP">
         <xforms:input ref="ZIP">
            <xforms:label>Zip Code</xforms:label>
      </field>
   </xforms:repeat>
```

Note: For detailed information about creating specific XFDL items, see the specific sections that describes how to create them, such as "Creating single line fields" on page 35 or "Creating lists" on page 46.

Example of completed table

The following code shows a sample of a completed table:

```
<table sid="T1">
   <xforms:repeat ref="person">
      <field sid="Name">
         <xforms:input ref="name">
            <xforms:label>Name</xforms:label>
      </field>
      <field sid="Street">
         <xforms:input ref="street">
            <xforms:label>Street</xforms:label>
      </field>
      <field sid="City">
         <xforms:input ref="city">
            <xforms:label>City</xforms:label>
      </field>
      <field sid="State">
         <xforms:input ref="State">
            <xforms:label>State</xforms:label>
      </field>
      <field sid="ZIP">
         <xforms:input ref="zip">
            <xforms:label>Zip Code</xforms:label>
      </field>
   </xforms:repeat>
   <fontcolor>white</fontcolor>
   <bgcolor>cadet blue</bgcolor>
   <itemlocation>
      <x>0</x>
      <y>23</y>
   </itemlocation>
   <size>
      <width>200</width>
      <height>100</height>
   </size>
</table>
```

Adding help messages

If you want to add help and accessibility messages to your forms, you have three XForms options:

- Hint
- Help
- Alert

You can also choose to continue to use the following XFDL options to provide assistance to your users:

- Help

- Acclabel
- Message (element of the *format* option)

XForms help messages allow you to use dynamic data from your instance node to create additional help or warning messages for your forms. You can also add static help messages between the control's tags. The *hint*, *help*, and *alert* controls must always be contained inside other form controls, such as *input*, *textarea*, and *select1*.

Adding a help message

The *help* control is optional, and allows you to provide a help message that is displayed to the user if they enter help mode. *Help* is intended to provide detailed help to the user.

Although there is no direct equivalent in XFDL, the *help* control is treated as like the *help* option, and is displayed as a tooltip when the user enters help mode.

If an item contains both a *hint* control and a *help* control, then *help* is appended to *hint*. Furthermore, if an item contains both a *help* control and an XFDL *help* option, then the *help* option overrides the *help* control.

The format of the *help* control is:

```
<xforms:help attribute></xforms:help>
```

Help has only one optional attribute:

- **ref** — The path to the data node that contains the help message. The format of *ref* is:

  ```
  ref="path_to_node"
  ```

  For example, if the *help* control was associated with the <address> node inside the help instance, then the XPath reference would be:

  ```
  ref="instance('help')/address"
  ```

  The *ref* attribute is optional. If *help* does not have a *ref* attribute, then the *help* control must contain a help message between its start and end tags. For example:

  ```
  <xforms:help>This is a help message.</xforms:help>
  ```

Help is always used inside other form controls that accept user input. For example:

```
<xforms:input ref="instance('personal_info')/address>
   <xforms:help>Type your address here.</xforms:help>
</xforms:input>
```

Help can be used with the following form controls:

- input
- output
- secret
- select
- select1
- submit
- textarea
- trigger
- upload

Hint

The *hint* control is optional, and allows you to provide a help message that is displayed to the user if they enter help mode. This message is generally a short instruction, such as telling the user what format is valid for a specific field, and is displayed as a tooltip.

This is equivalent to the XFDL *help* option. Furthermore, if an item contains both a *hint* control and an XFDL *help* option, then the *help* option overrides the *hint* control.

Note: If an item contains both a hint control and a help control, then help is appended to hint.

The format of the *hint* control is:

```
<xforms:hint attribute></xforms:hint>
```

Hint has only one optional attribute:

- **ref —** The path to the data node that contains the hint. The format of *ref* is:

  ```
  ref="path_to_node"
  ```

  For example, if the *hint* control was associated with the <address> node inside the help instance, then the XPath reference would be:

  ```
  ref="instance('help')/address"
  ```

  The *ref* attribute is optional. If *hint* does not have a *ref* attribute, then the *hint* control must contain a hint between its start and end tags. For example:

  ```
  <xforms:hint>This is a hint.</xforms:hint>
  ```

Hint is always used inside other form controls that accept user input. For example:

```
<xforms:input ref="instance('personal_info')/address>
   <xforms:hint>Type your address here.</xforms:hint>
</xforms:input>
```

Hint can be used with the following form controls:

- input
- output
- secret
- select
- select1
- submit
- textarea
- trigger
- upload

Alert

The *alert* control is optional, and allows you set an alert message that is displayed to the user if they enter invalid information. This is equivalent to the *message* setting in the *format* option. If both an *xforms:alert* and a *message* are provided for an item, then the *message* overrides the *xforms:alert*.

The format of the *alert* control is:

```
<xforms:alert attribute></xforms:alert>
```

Alert has only one optional attribute:

- **ref —** The path to the data node that contains the alert. The format of *ref* is:

  ```
  ref="path_to_node"
  ```

  For example, if the *alert* control was associated with the <alert> node inside the help instance, then the XPath reference would be:

  ```
  ref="instance('help')/alert"
  ```

  Ref is optional. If *alert* does not have a *ref* attribute, then the *alert* control must contain a warning between its start and end tags. For example:

  ```
  <xforms:alert>Mandatory field. Please type your name.</xforms:hint>
  ```

Alert is always used inside other form controls that accept user input. For example:

```
<xforms:input ref="instance('personal_info')/address>
   <xforms:alert>Mandatory field. Please type your name.</xforms:hint>
</xforms:input>
```

Alert can be used with the following form controls:

- input
- output
- secret
- select
- select1
- submit
- textarea
- trigger
- upload

Adding actions

XForms actions define specific actions in the form. For example, *insert* allows you to add a new row to a table, while *message* allows you to add popup messages to your form. Actions are triggered by *events*. Events may be user initiated, such as **DOMActivate** or application initiated, like **xforms-ready**.

All XForms actions are in the XForms namespace, which means that every action must be prefixed by xforms:, as shown:

```
xforms:action
```

User initiated actions are placed inside the *trigger* control, as shown:

```
<button sid="Button1">
   <xforms:trigger>
      ... action ...
   </xforms:trigger>
</button>
```

If you want to have an action triggered by an automatic event inside the form, you must place the action inside the XFDL item that the action will effect. For example:

```
<field sid="FIELD1">
   <xforms:message
      level="modal"
      ev:event="xforms-ready">Format: (###) ###-####</xforms:message>
   ...xforms:input...
   ...XFDL options...
</field>
```

All XForms actions are modified by attributes, such as XPath references or nodesets and event details. This information is added to actions as attributes, as shown:

```
<...action... attributes></...action...>
```

There are 13 XForms actions:

• delete	• insert
• message	• rebuild
• recalculate	• refresh
• reset	• revalidate
• send	• setfocus
• setindex	• setvalue
• toggle	

Delete

The *delete* action deletes specified nodes from the instance data. If you delete a parent node, its children are deleted as well.

The format of the *delete* action is:

```
<xforms:delete ...attributes.../>
```

Delete has three attributes:

- **at** — Indicates the location of the item (in the form display) that will be deleted.
- **nodeset** — Indicates the node or nodeset to be deleted.
- **event** — Indicates the type of event that is triggered. In the case of *delete*, it is usually **DOMActivate**.

For example:

```
<xforms:delete
   at="index('item_sid')"
   nodeset="node"
   ev:event="DOMActivate">
</xforms:delete>
```

Delete is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *delete* action inside the *trigger* control:

```
<xforms:trigger>
   <xforms:label>Delete a row</xforms:label>
   <xforms:delete
      at="index('customer_table')"
      nodeset="customer"
      ev:event="DOMActivate"/>
</xforms:trigger>
```

Insert

The *insert* action adds new nodes to a set of nodes that all have the same names and attributes, although their values differ. *Insert* adds the new node by cloning the final node in a nodeset.

The format of the *insert* action is:

```
<xforms:insert ...attributes.../>
```

Insert has four attributes:

- **at —** Determines where the new items will be inserted. This attribute usually refers to an XFDL group-type item, such as *pane, table,* or *group*.
- **position —** Indicates whether the inserted node is added before or after the anchor (referenced) node.
- **nodeset —** Indicates the node or nodeset to be deleted.
- **event —** Indicates the type of event that is triggered. In the case of *insert*, it is usually **DOMActivate**.

For example:

```
<xforms:insert
   at="index('item_sid')"
   position="before_or_after"
   nodeset="node"
   ev:event="DOMActivate"/>
```

Insert is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *insert* action inside the *trigger* control:

```
<xforms:trigger>
   <xforms:label>Add a row</xforms:label>
   <xforms:insert
      at="index('customer_table')"
      position="after"
      nodeset="customer"
      ev:event="DOMActivate"/>
</xforms:trigger>
```

Message

The *message* action displays a message to the user.

The format of the *message* action is:

```
<xforms:message ...attributes...>...message text...</xforms:message>
```

Message has two attributes:

- **event —** Indicates the type of event that is triggered. In the case of delete, it is usually **DOMActivate**.
- **level —** Indicates the type of message display. The possible settings of this attribute are:
  - **ephemeral —** A popup message that lasts only as long as the related item has the focus.
  - **modeless —** A popup message that does not interfere with the user’s access to the form. Users have the option of closing the message.
  - **modal —** A popup message that users must close before they can proceed with the form.

For example:

```
<xforms:message
   level="level"
   ev:event="DOMActivate">...message text...</xforms:message>
```

Message is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *message* action inside an XFDL field:

```
<field sid="FIELD1">
   <xforms:message
      level="modal"
      ev:event="xforms-ready">Format: (###) ###-####</xforms:message>
   ...xforms:input...
   ...XFDL options...
</field>
```

Rebuilding actions

The *rebuild*, *recalculate*, *refresh*, and *revalidate* actions trigger rebuilds of the form, including calculations and other computes. *Rebuild* rebuilds the entire form, *recalculate* recalculates computes and other calculations, *refresh* refreshes the form, and *revalidate* revalidates it against the XForms schema. Under normal operating conditions, these actions take place automatically without the need for a manual trigger and form authors do not need to take them into consideration. However, if you are building forms in a limited environment where controlling these actions is more desirable than allowing them to occur automatically, you may need to use call them manually.

The format of the rebuilding actions is:

```
<xforms:<action> ...attributes.../>
```

They have the following attributes:

- **model —** The id of the model you want to rebuild. This attribute is optional. If a model is not specified, the default model is used.
- **event —** Indicates the type of event that is triggered. To trigger a manual rebuild, it is usually **DOMActivate**.

For example:

```
<xforms:rebuild
   model="id"
   ev:event="DOMActivate"/>
```

The rebuilding actions are always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *rebuild* action inside the *trigger* control:

```
<xforms:trigger>
   <xforms:rebuild
      model="loan"
      ev:event="DOMActivate"></xforms:rebuild>
   ...xforms:label...
</xforms:trigger>
```

Reset

The *reset* action resets the form back to its original opening state.

The format of the *reset* action is:

```
<xforms:reset ...attributes.../>
```

Reset has the following attributes:

- **model** — The id of the model you want to reset. This attribute is optional. If a model is not specified, the default model is used.
- **event** — Indicates the type of event that is triggered. To trigger a manual reset, it is usually **DOMActivate**.

For example:

```
<xforms:reset
   model="id"
   ev:event="DOMActivate"/>
```

Reset is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *reset* action inside the *trigger* control:

```
<xforms:trigger>
   <xforms:reset
      model="loan"
      ev:event="DOMActivate"/>
   ...xforms:label...
</xforms:trigger>
```

Send

The *send* action transmits data, following rules defined by the submission that determine what data is submitted, how the data is submitted, and where the data goes. The *send* and *submit* actions perform similar duties; however, *send* can also be used as part of a *trigger* control that includes multiple actions.

The format of the *send* action is:

```
<xforms:send ...attributes.../>
```

Send has two attributes:

- **submission** — The id of the submission you want to send.
- **event** — Indicates the type of event that is triggered. In the case of *send*, it is always **DOMActivate**.

For example:

```
<xforms:send
   submission="id"
   ev:event="DOMActivate"/>
```

Send is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *send* action inside the *trigger* control:

```
<xforms:trigger>
   <xforms:send
      submission="SubmitLoanData"
      ev:event="DOMActivate"/>
   ...xforms:label...
</xforms:trigger>
```

Setfocus

The *setfocus* action redirects the focus from the current form control to a different form control. This action is used to improve accessibility for users with disabilities.

The format of the *setfocus* action is:

```
<xforms:setfocus ...attributes.../>
```

Setfocus has the following attributes:

- **control —** Indicates which form control gets the focus.
- **event —** Indicates the type of event that is triggered. In the case of *setfocus*, it is always **DOMActivate**.

For example:

```
<xforms:setfocus
   contol="id"
   ev:event="DOMActivate"/>
```

Setfocus is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *setfocus* action inside the *trigger* control:

```
<xforms:toggle>
   <xforms:setfocus
      control="./loan_request_help"
      ev:event="DOMActivate"/>
</xforms:toggle>
```

Note: A known limitation of setfocus in XForms 1.0 is that it cannot identify controls generated by a repeat table.

Setindex

The *setindex* action is similar to the *setfocus* action as it allows you to specify the position of the focus. The difference is that *setindex* allows you to specify the position of the focus in the *index*. The index is the method used by each *repeat* control to keeps track of which item of the form's *repeat* controls currently has the focus. In other words, *setindex* allows you to specify the position of the focus in the index, while *setfocus* allows you to specify the position of the focus for individual form controls.

The format of the *setindex* action is:

```
<xforms:setindex...attributes.../>
```

Setindex has the following attributes:

- **repeat —** The *id* of the *repeat* control.
- **index** — The number of the row that has the focus. For example, if the focus was on the third row of the specified table, then this value would be **3**.
- **event —** Indicates the type of event that is triggered. In the case of *setfocus*, it is always **DOMActivate**.

For example:

```
<xforms:setindex
   repeat="id"
   index="row_number"
   ev:event="DOMActivate"/>
```

Setindex is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *setindex* action inside the *trigger* control:

```
<xforms:trigger>
   <xforms:setindex
      repeat="loan_table"
      index="3"
      ev:event="DOMActivate"/>
      ...xforms:label...
</xforms:trigger>
```

Note: A known limitation of setindex in XForms 1.0 is that it cannot identify controls generated by a repeat table.

Setvalue

The *setvalu*e action sets the value of the specified instance node.

The format of the *setvalue* action is:

```
<xforms:setvalue ...attribute.../>
</xforms:setvalue>
```

Setvalue has the following attributes:

- **value** — An XPath expression that evaluates to the desired value.
- **event** — Indicates the type of event that is triggered. In the case of *setfocus*, it is usually **xforms-ready**. For example:

  ```
  <xforms:setindex
     value="XPath_Ref"
     ev:event="xforms-ready"/>
  ```

Setvalue also supports literal values. To set a literal value, you must remove the *value* attribute, as literal values and XPath references are mutually exclusive in *setvalue*. For example:

```
<xforms:setindex ev:event="xforms-ready">Item Number</xforms:setindex>
```

Setvalue is always connected with an event that activates the action. The event may be activated by the user, such as a button triggering the *DOMActivate* event, or it may be an application controlled event, such as *xforms-ready*, which would be placed inside the XFDL item the action will effect. The following example shows the *setvalue* action inside the XFDL item the action will effect:

```
<field sid="Field1>
   <xforms:setvalue ev:event="xforms-ready">Item Number</xforms:setvalue>
   ...xforms:input...
   ...XFDL options...
</field>
```

Toggle

The *toggle* action selects one possible case from all the possible cases contained in a *switch* group. In other words, *toggle* ensures that the selected case is turned on, while the other cases are turned off.

The format of *toggle* is:

```
<xforms:toggle ...attributes.../>
```

Toggle has two attributes:

- **event** — Indicates the type of event that is triggered. This attribute is mandatory and is written as:

  ```
  ev:event="event"
  ```

In the case of *toggle*, the event is usually **DOMActivate**.

- **case —** Indicates which case is selected when the Toggle button is pushed. The format of *case* is:

  ```
  case="name"
  ```

  This attribute must match the *id* of the *case* control you want to select. For example, if you wanted a case with an *id* of "Spanish" to be selected, you would write:

  ```
  case="Spanish"
  ```

The *toggle* action is usually nested inside the *trigger* control. For example:

```
<button sid="SwitchToSpanish">
   <xforms:trigger>
      <xforms:toggle
         ev:event="DOMActivate"
         case="Spanish"/>
      ...xforms:label...
   </xforms:trigger>
   ...XFDL options...
</button>
```

Adding multiple actions to a form control

You can't add multiple actions to a form control unless they are contained inside the *action* action. *Action* groups actions to form a top-down sequence.

The format of *action* is:

```
<xforms:action ...attributes...>
   ...action1...
   ...action2...
   ...action3...
</xforms:action>
```

When *action* is used to group actions, it contains the *ev:event* attribute for the group. Any *ev:event* attributes that are normally part of the component actions are dropped.

For example:

```
<xforms:action
   ev:event="DOMActivate">
   <xforms:insert
      at="index('item_sid')"
      position="before_or_after"
      nodeset="node"/>
   <xforms:message
      level="mode"> Enter the purchase order number.</xforms:message>
</xforms:action>
```

Like other actions, the *action* group must be contained in a form control, such as *trigger*, which activates the actions. For example:

```
<xforms:trigger>
   <xforms:label>Add a row</xforms:label>
   <xforms:action
      ev:event="DOMActivate">
      <xforms:message level="modeless"
         >Enter the purchase order number and quantity.</xforms:message>
      <xforms:insert
         at="index('customer_table')"
```

```
         position="after"
         nodeset="customer/"
   </xforms:action>
</xforms:trigger>
```

About events

When you create a button, you create an item that allows users to initiate an event. The initiation of this event activates the associated action. However, sometimes you may want to initiate an action because of something that has happened in the form, without user intervention. This is called an application initiated event. When these events occur, they trigger any action that lists that event as an attribute.

While there is only one user initiated event (DOMActivate), there are application initiated events:

- **xforms-ready —** Indicates that all automatic events have occurred and the form is ready for user input or additional processing.
- **xforms-model-construct** — Initiates the construction of the XForms model.
- **xforms-model-construct-done —** Indicates that the construction of the XForms model is complete.
- **xforms-model-destruct** — Indicates that the form has been closed by the user.

For detailed information about events, see the *Workplace Forms XFDL Specification* document.

Creating submission rules for an instance

When submitting a form that contains an XForms data model, you can submit either the entire form or just a particular data instance. This makes it possible to send your data instance directly to processing applications, rather than having to parse the complete form and extract the data instance.

If you want to submit a data instance, you must create a set of submission rules. These rules help determine what data is submitted, how the data is submitted, and where the data goes. In addition to submission rules, you must also create a submission button that is linked to the rules. For detailed information about creating a submission button, see “Creating a submission button” on page 87.

Each set of submission rules is inserted within the <xforms:submission> tag in the data model, as shown:

```
<xforms:model>
   <xforms:submission ...attributes...>
   </xforms:submission>
</xforms:model>
```

Each submission is further defined by adding attributes to the <submission> tag and by including optional serialization rules. This is explained in more detail in the following sections.

Naming the submission rules

Each submission tag must include an *id* attribute. This attribute names the submission rules, allowing the submission to be identified and triggered by the submit button. Each *id* should be unique. The *id* attribute follows this format:

```
id="name"
```

For example, if you wanted to call the submission rules *SubmitLoanData*, you would use the following tag:

```
<xforms:submission id="SubmitLoanData">
```

Setting the type of submission

Each submission tag must include the *method* attribute. This attribute describes how the submission is performed.

The *method* attribute can have either of two values:

- **post** — Serializes the data and sends it as XML.
- **get** — Serializes the data and sends it as URL encoded data.
- **put** — Serializes the data and saves it as a file instead of submitting it.

For example, if you wanted to submit the data in XML, you would use the following tag:

```
<xforms:submission id="SubmitLoanData"
   method="post"/>
```

Setting the target URL for a submission

Use the *action* attribute to define the target URL for the submission. The *action* attribute is written in the following format:

```
action="URL"
```

You can only list one URL in the *action* attribute. For example, if you wanted to submit your data instance to a cgi script on your server, you might use the following submission tag:

```
<xforms:submission id="SubmitLoanData"
   action="http://www.myserver.com/cgi">
```

Note: The Viewer supports both HTTP: and HTTPS: protocols.

If you are using the *put* method and want to save your submission as a file, you must use the *file:* scheme. This indicates that the directory and file in which the submission should be saved. For example:

```
<xforms:submission id="SubmitLoanData"
   action="file:\\\C:\Documents and Settings\curchen\xforms\PO\instance">
```

Note: There are some limitations to where you may save submission data. A *put* submission will fail if you try to save it to the following locations:

- The *Program Files* directory.
- The system drive, the *Windows* directory, or the *Windows System* directory.
- Temporary directories.
- A directory outside the folder subtree containing the originating file.

Setting the content type of the submission

Your submission rules may also include an optional *mediatype* attribute that declares the content type for posted data. This attribute does not change the actual content type of the submission; it simply declares the document's content type in the submission header.

For example, if you wanted to set a MIME type of *application/vnd.xfdl* you would use the following tag:

```
<xforms:submission id="SubmitLoanData"
   mediatype="application/vnd.xfdl"/>
```

If you do not provide a media type, it defaults to *application/xml*.

Note: If you do set the submission's *mediatype* attribute, you must ensure that the submitted data conforms to the syntax of the specified *mediatype*. If the submitted data does not match the *mediatype*, there may be submission errors.

Setting what data is submitted

By default, the first data instance in a form is submitted. If you want to submit something other than the entire first data instance, you must provide one of the following:

- An XPath expression that specifies a node (and its children) for submission.
- The name of a bind that references the nodeset you want to submit.

Specifying nodes

You can choose to submit an entire data instance or only a portion of the instance. When submitting only a portion of a data instance, you must identify the *root element* of the submission. The root element determines which portion of the instance is submitted, since only the root element and its children are sent.

This is done using the *ref* attribute:

```
ref="root_node_of_the_submission"
```

For example, consider the following data instance:

```
<xforms:instance xmlns="" id="loan">
   <LoanRecord>
         <StartDate></StartDate>
      <Borrower>
         <Name>John Q. Public</Name>
         <Addr>123 Main St. Tinyville</Addr>
    </Borrower>
      <Principal currency="USD"></Principal>
      <Duration xsi:type="xsd:positiveInteger"></Duration>
      <InterestRate xsi:type="xsd:integer"></InterestRate>
      <MonthlyPayment></MonthlyPayment>
      <TotalPayout></TotalPayout>
   </LoanRecord>
</xforms:instance>
```

If you wanted to submit the entire instance, you would need to select the *root node* of the instance. By default, the first tag in an instance is its root node. In the above case, the root node would be <LoanRecord>:

```
<xforms:submission id="SubmitPersonalData"
   ref="instance('loan')>
```

If you wanted to select only a portion of the instance, you would have to select the root node of the data you wanted to submit. For example, if you only wanted to submit the <Borrower> information, you would reference the <Borrower> node. Specifying a parent node includes all of its child nodes in the submission. In the following example, the *ref* attribute indicates that the submission data should consist of the <Borrower> node and its child nodes, <Name> and <Address>:

```
<xforms:submission id="SubmitPersonalData"
   ref="instance('loan')/Borrower">
```

Ref also lets you use an XPath expression that computes the node that will be submitted. The default value of *ref* is / (forward slash).

Specifying a bind

Another way of stipulating which data you want to submit is by referencing a bind. By referencing a bind, you specify the bind's nodeset as the data you want to submit. Furthermore, all of the limitations you placed on the bind are relevant to the submission data. For example, if the bind indicates that certain of its nodes are not valid, then those nodes would not be included in the submission, even though they were part of the indicated nodeset.

This is done using the *bind* attribute:

```
bind="bind_id"
```

You can specify a particular bind by referring to its *id* attribute. For example, if the *id* attribute of a bind is ″loan_history″, then the submissions's *bind* attribute should contain ″loan_history″ as its setting, as shown:

```
<xforms:submission
   id="SubmitPersonalData"
   bind="loan_history"/>
```

Note: Remember, the submission id is the name of the submission. The bind id is the name of the bind that contains the nodeset you want to submit.

Filtering inherited namespaces

By default, when you submit a data instance, the instance includes all of the namespaces that it inherits. For example, consider the following XForms model:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <XFDL xmlns="http://www.PureEdge.com/XFDL/7.0"
   XFDL xmlns:xforms="http://www.w3.org/2002/xforms"
   xmlns:custom="http://www.PureEdge.com/XFDL/Custom">
   ...
   <xformsmodels>
      <xforms:model xmlns:xforms="http://www.w3.org/2002/xforms">
         <xforms:instance xmlns="http://www.mycompany.com/loans"
             id="loan">
            <LoanRecord>
               <StartDate></StartDate>
               <Borrower>
                  <Name>John Q. Public</Name>
                  <Addr>123 Main St. Tinyville</Addr>
               </Borrower>
               <Principal currency="USD"></Principal>
               <MonthlyPayment></MonthlyPayment>
               <TotalPayout></TotalPayout>
            </LoanRecord>
         </xforms:instance>
         ...
   </xformsmodels>
   ...
   </XFDL>
```

When you submit the *loan* instance, the root element of the submission is modified so that it declares all of the namespaces that it inherits. Assume that the root element is *LoanRecord*. In this case, that tag declares a default namespace but also inherits the XFDL, custom, and XForms namespaces from the XFDL tag. As a result, the submission declares those namespaces on the *LoanRecord* element, as shown:

```
<LoanRecord xmlns="http://www.mycompany.com/loans"
   xmlns="http://www.PureEdge.com/XFDL/7.0"
   XFDL xmlns:xforms="http://www.w3.org/2002/xforms"
   xmlns:custom="http://www.PureEdge.com/XFDL/Custom">
```

In some cases, you may want to restrict the inherited namespaces that are included with a data instance. For example, you may want to submit non-namespace-aware XML for DTD validation.

To restrict the inherited namespaces that are included, use the *includenamespaceprefixes* attribute. This attribute lists the prefixes for those namespaces that you want to include in the submission, and follows this syntax:

```
includenamespaceprefixes="prefix1 prefix2 prefix3"
```

Each prefix is separated by whitespace, such as a space. For example, to include only the XFDL and the custom namespaces in a submission, you would use the following submission tag:

```
<xforms:submission id="SubmitLoanData"
   includenamespaceprefixes="XFDL custom">
```

If you want to submit only those namespaces that are used in your data instance, you can do this automatically by using an empty string, as shown:

```
<xforms: submission id="SubmitLoanData"
   includenamespaceprefixes="">
```

This automatically removes all namespaces that are not used in your data instance.

If you want to include only the default namespace, there is a special value you must use: **#default**. For example:

```
<xforms:submission id="SubmitLoanData"
   includenamespaceprefixes="#default">
```

This ensures that only the form's default namespace is included in the submission.

Displaying data returned by the submission

By default, when data is returned by the submission, it replaces the submitted form. For example, when a user submits a form, the server often returns a reply that indicates that the submission was received. Generally, this notice entirely replaces the submitted form.

However, XForms lets you handle returned data more flexibly. Instead of simply replacing the entire form, you can choose to replace a single instance or to even ignore the returned data. This is done using the *replace* attribute. This attribute indicates whether the returned data should replace the entire form, a data instance, or be ignored.

Replace has three possible options:

- **all** — The returned data replaces the entire form.
- **instance** — The returned data only replaces the submitted instance.
- **none** — The returned data is ignored.

The format of this attribute is:

```
replace="return_option"
```

So, if you wanted to replace the submitted data instance instead of the whole form, you would use the following submission tag:

```
<xforms:submission id="SubmitLoanData"
   replace="instance">
```

Creating a submission button

Submission buttons are necessary to submit the instance data to a processing server. The button triggers the submission, and the data submitted is determined by combining the filters for the button with the submission rules in the data model.

To create a submission button, create a button with a form control of *submit* or *trigger.*

```
<button sid="Submit">
   <xforms:submit ...attributes...>
   </xforms:submit>
</button>
```

To link the submit button to the appropriate submission rules, you must refer to the name of the submission. This name is defined by the *id* attribute of the appropriate *submission* tag.

For example, if your submission rules had an *id* of ″SubmitLoanData″, you would use the following button:

```
<button sid="Submit">
   <xforms:submit submission="SubmitLoanData">
   </xforms:submit>
</button>
```

A button with a form control of *trigger* performs essentially the same action. However, it requires you to specify information that is assumed by the *submit* form control. For example, if you choose to use *trigger* to create a submit button, *trigger* must enclose the *send* action and its attributes, as shown:

```
<button sid="Submit2">
   <xforms:trigger>
      <xforms:send
         ev:event="DOMActivate" submission="SubmitLoanData">
      </xforms:send>
   </xforms:trigger>
</button>
```

Note: Both the submit and trigger form controls have mandatory nested form controls, such as label. For detailed information on these controls, see “Creating user interface links” on page 21.

Adding schemas

When adding schemas to your form, you can choose to either embed one or more schema files in the form itself, or refer to external schema files that are saved on the user's computer. Embedding schemas in the form will increase the overall size of the form and may affect performance, especially when low bandwidth is available. However, referring to external schema files requires you to distribute those files to client computers. The architecture of your overall application will probably dictate which solution you should use.

Normally, each instance in the data model is validated against all available schemas. However, if a schema is defined for a particular namespace, only those instances that belong to that namespace are validated against it. This allows you to apply specific schemas to specific data instances.

Note: Unlike the XML Data Model, XForms automatically performs schema validation. Therefore, you do not need to use XFDL functions to manually validate the schema.

Finally, you must restrict all schemas to a single, self-contained file. The Workplace Forms Viewer does not support the use of the *import* or *include* tags.

Embedding a schema in a form

You can embed any number of schemas in a form. Each schema is inserted in its own <xsd:schema> tag in the XForms model, as shown:

```
<xforms:model>
   <xsd:schema>
      ... schema ...
   </xsd:schema>
</xforms:model>
```

Each schema is defined in a separate <xsd:schema> tag, as shown:

```
<xforms:model>
   <xsd:schema>
      ... schema 1 ...
   </xsd:schema>
   <xsd:schema>
      ... schema 2 ...
   </xsd:schema>
</xforms:model>
```

Each schema is placed between the opening and closing schema tags, and must conform to the rules for XML schemas.

Referring to external schemas

External schemas must be placed in the Viewer's *schema* folder or they will not be available to the form. You can add sub-folders to the Viewer's *schema* folder, but cannot place any schemas outside of that folder. Only those external schemas listed in the *schema* attribute on the <xforms:model> tag are used during validation. Any other schemas in the Viewer's *schema* folder are ignored.

The *schema* attribute is written as shown:

```
schema="list of schemas"
```

You must list each of the external schemas by path and filename, relative to the Viewer's *schema* folder. This list is space delimited, which means that your schema filenames cannot contain spaces. Furthermore, you must add a prefix of *xsf:* to each path. For example, if both the *personalData.xsd* and *rateData.xsd* schemas we in the Viewer's *schema* folder, you would use the following *schema* attribute to register those schemas:

```
schema="xsf:personalData.xsd xsf:rateData.xsd"
```

The *schema* attribute is added to the <xforms:model> tag. For example, a complete data model, with external *personalData* and *rateData* schemas, would look like this:

```
<xforms:model schema="xsf:personalData.xsd xsf:rateData.xsd">
</xforms:model>
```

Sample XForms

The following pages show a complete sample of the core XForms data model, complete with bindings and submissions. There is also a sample of a form with a schema. For samples of UI links, such as form controls and actions, see Appendix XXXX.

Core XForms data model

The following example shows a core XML Data Model with three data instances. This data model includes submission rules that are linked to the *Submit* button in the form. These submission rules send the entire form to the back-end for processing.

```
<xformsmodels>
   <xforms:model functions="power">

      <xforms:instance xmlns="" id="loan">
         <LoanRecord>
            <StartDate>2004-12-02</StartDate>
            <Borrower>
               <Name>John Q. Public</Name>
               <Addr>123 Main St. Tinyville</Addr>
            </Borrower>
            <Principal currency="USD"></Principal>
            <Duration xsi:type="xsd:positiveInteger"></Duration>
            <InterestRate xsi:type="xsd:integer"></InterestRate>
            <MonthlyPayment></MonthlyPayment>
            <TotalPayout></TotalPayout>
         </LoanRecord>
      </xforms:instance>

      <xforms:instance xmlns="" id="currencyChoices">
         <data>
            <choice show="US Dollars">USD</choice>
            <choice show="CDN Dollars">CDN</choice>
            <choice show="Euros">Euro</choice>
         </data>
      </xforms:instance>

      <xforms:instance xmlns="" id="rate">
         <rate>NaN</rate>
      </xforms:instance>

      <xforms:bind
         calculate="instance('loan')/InterestRate div 1200.0"
         nodeset="instance('rate')">
      </xforms:bind>

      <xforms:bind
         nodeset="Borrower/*"
         required="true()">
      </xforms:bind>

      <xforms:bind
         nodeset="Principal"
         required="true()">
      </xforms:bind>

      <xforms:bind
         nodeset="StartDate"
         type="xsd:date">
      </xforms:bind>

      <xforms:bind
         calculate="../MonthlyPayment * ../Duration"
         nodeset="TotalPayout"
         relevant="../MonthlyPayment > 0 and ../Duration > 0">
      </xforms:bind>

      <xforms:submission id="SubmitLoanData"
         action="http://igor:5005/cgi-bin/doReturn-full"
         includenamespaceprefixes=""
         method="post">
      </xforms:submission>

   </xforms:model>
</xformsmodels>
```

Data model with schema validation

The following data model contains all the core parts of a data model, as well as both an external schema and an internal schema.

```
<xformsmodels>
   <xforms:model schema="xsf:personalData.xsd">

      <xforms:instance xmlns="" id="loan">
         <LoanRecord>
            <StartDate>2004-12-02</StartDate>
            <Borrower>
               <Name>John Q. Public</Name>
               <Addr>123 Main St. Tinyville</Addr>
            </Borrower>
            <Principal currency="USD"></Principal>
            <Duration xsi:type="xsd:positiveInteger"></Duration>
            <InterestRate xsi:type="xsd:integer"></InterestRate>
            <MonthlyPayment></MonthlyPayment>
            <TotalPayout></TotalPayout>
         </LoanRecord>
      </xforms:instance>

      <xforms:instance xmlns="" id="currencyChoices">
         <data>
            <choice show="US Dollars">USD</choice>
            <choice show="CDN Dollars">CDN</choice>
            <choice show="Euros">Euro</choice>
         </data>
      </xforms:instance>

       <xforms:instance xmlns="" id="rate">
         <rate>NaN</rate>
       </xforms:instance>

      <xsd:schema>
         ... schema details ...
      </xsd:schema>

      <xforms:bind
         calculate="instance('loan')/InterestRate div 1200.0"
         nodeset="instance('rate')">
      </xforms:bind>

      <xforms:bind
         nodeset="Borrower/*"
         required="true()">
      </xforms:bind>
      <xforms:bind
         nodeset="Principal"
         required="true()">
      </xforms:bind>

      <xforms:bind
         nodeset="StartDate"
         type="xsd:date">
      </xforms:bind>

      <xforms:bind
         calculate="../MonthlyPayment * ../Duration"
         nodeset="TotalPayout"
         relevant="../MonthlyPayment > 0 and ../Duration > 0">
      </xforms:bind>

      <xforms:submission id="SubmitLoanData"
         action="http://igor:5005/cgi-bin/doReturn-full"
         includenamespaceprefixes=""
         method="post">
      </xforms:submission>

   </xforms:model>
</xformsmodels>
```

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Office 4360
One Rogers Street
Cambridge, MA 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX
IBM
Workplace
Workplace Forms

Other company, product, or service names may be trademarks or service marks of others.

Index

Program Number:

Printed in USA