XForms and OpenDocument in OpenOffice.org

J. David Eisenberg

Let's say you're in charge of a database of chartered clubs for an amateur sports association. Club directors send you papers like the one in Appendix A, and you enter the data into an XML file, which is used to create an online searchable database of the clubs. Clearly, the better option is to send the club directors a machine-readable document with form fields that they fill in. The directors then click a button on the form and have the data sent to a server, so you don't have to decipher their handwriting and re-enter the data.

The idea of having a document with user-modifiable fields is not a new one, but OpenDocument's use of the World Wide Web Consortium's XForms recommendation (http://www.w3.org/MarkUp/Forms/) is especially noteworthy, because the data that is stored with the document is in XML format, not in some proprietary format dependent upon a single vendor's tools.

How XForms Works

A form in XForms is described by XML elements. Each form is composed of a *model* and *form controls*. The model contains

- Instance data: an XML "template" that will be filled in by information in the form controls.
- Submission: a description of where the instance data is to be sent and how it is to be sent. A form can have multiple ways of being submitted; one to save the data into a file, one to send to a web URL, another to send to a different URL.
- Binding: a connection between a node in the instance data and a form control (text box, drop-down menu, etc), or a connection between instance data and a constraint in a model item.

XForms is not a stand-alone system; instead, you put the XForms markup into a document and hand the document to an application such as OpenOffice.org that is XForms aware. Under the guidance of the bindings, the application handles all the interaction between the controls (which the application "owns") and the instance data. In the fine tradition of separating content and presentation, XForms provides the logical structure; the application provides the presentation.

In this case, we're going to put the XForms markup into an OpenDocument word processing document, and hand it OpenOffice.org. Of course, we're not going to type all the XML elements by hand. Instead, we'll use the GUI in OpenOffice.org to create the form. The first step is to create a new XML forms document from the menu.

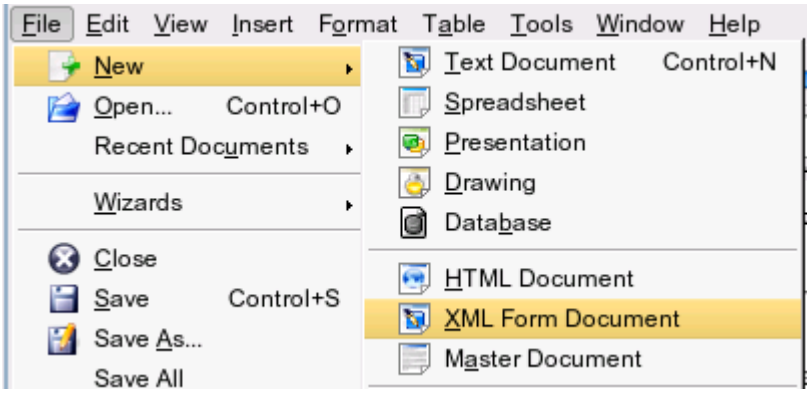

Illustration 1: Creating an XForms Document

When you create such a document, you'll see the parts of the XForms model on the screen:

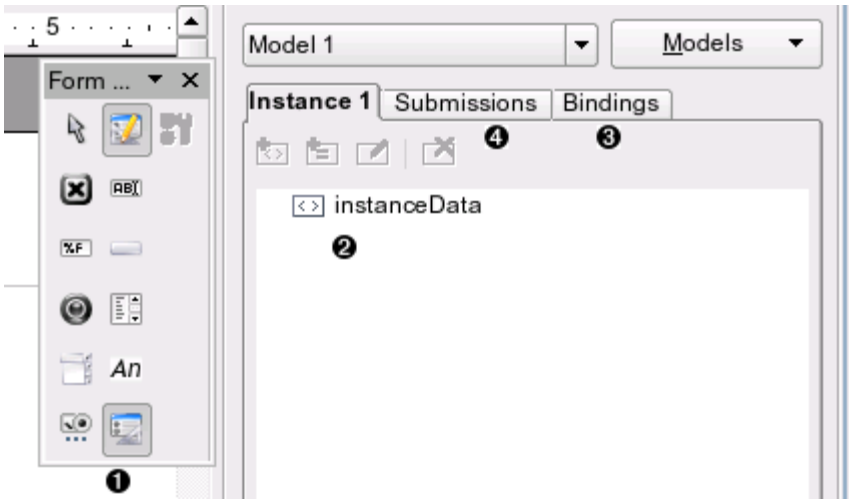

Illustration 2: GUI for OpenOffice.org's XForms builder

1. The form controls that go into your document. The form control icons are, in the first column: select a control, checkbox, formatted field, option button, combo box, and "more options." The icons in the second column are: design mode on/off, text box, pushbutton, list box, label field, and form design (turns toolbar on and off). The icon in the third column is the control tool, which brings up the properties dialog for the currently selected control).
2. The *instance data* is the XML that will hold the data that's entered into the form controls.
3. The *bindings* connect the form controls to the instance data.
4. The *submissions* tell how you want to send the instance data.

Text Areas

In order to test as we build, we'll build this document in parts. Let's start by creating the text area controls for the club name, contact person, phone, and email, and city. We'll handle the comments and date areas later. You use the text box tool (the second one in the second column in Illustration 2) to draw the areas. If you double-click on an area, you'll see a dialog pop up that lets you name the control. The "Label Field" is a way of associating a text box with a label control. In this case, we've simply put plain document text next to the text boxes rather than using a specific label field. You may want to give the fields a meaningful name, though it's not necessary. You can also change the font if you wish.

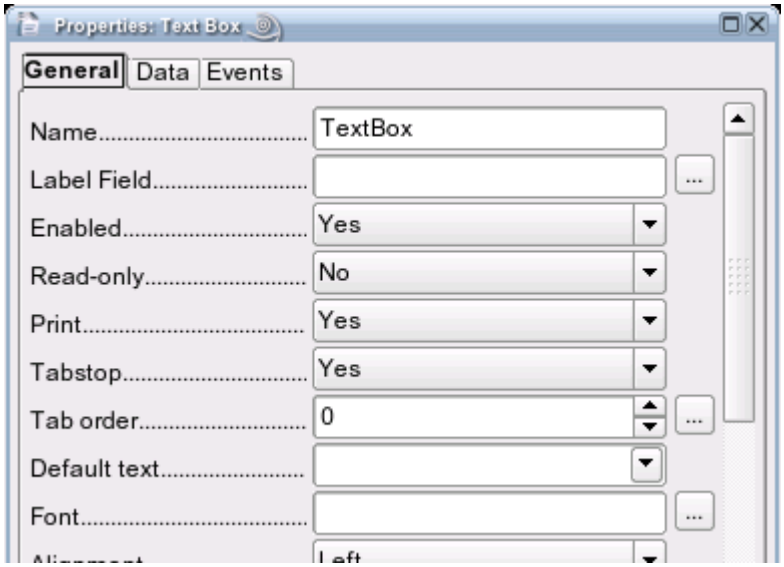

Illustration 3: Dialog for renaming text areas

Instance Data

Let's turn our attention to the instance data (the item numbered 2 in Illustration 2). It's named "Model 1" because a single document can have several forms, each with its own instance data, submissions, and bindings. We'll change the model name to `CAUSAW` (click the "Models" button and select "Rename"). Then we'll change the instance data name to `clubData`. When you change the instance name, you get the following dialog:

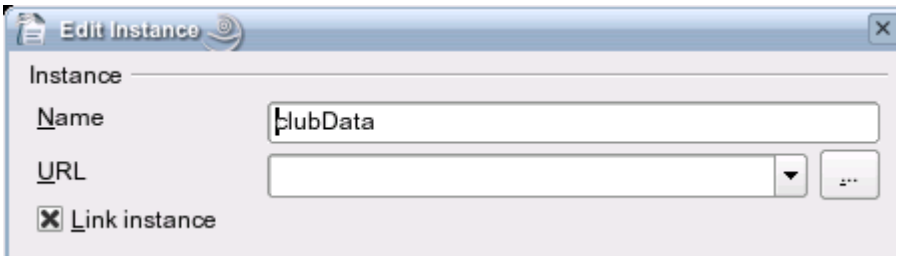

Illustration 4: Edit Instance dialog box

We want to have the data from the form fill in an XML template that looks like this. (Sample data is in italics.)

```
<club association="SCVWA" id="">
     <name>San Jose W-Squad</name>
     <contact>
          <name>Moishe Pipik</name>
          <email>moishe_pipik@example.com</email>
          <phone type="home">408-555-7871</phone>
     </contact>
     <city>San Jose, CA</city>
     <age kids="Y" cadet="Y" junior="Y" open="N"  women="Y"/>
     <comments>
          Practices Tue. and Thu. 6:00-8:00 p.m. at East H.S.
     </comments>
     <date>2006-07-31</date>
</club>
```

By using the icons in the instance data area (add an element, add an attribute, and edit an element or attribute), as shown in the following illustration, you can create the instance data "tree." When you add an element, it will be added as a child of the highlighted element. When you add an attribute, it will show up with a preceding @, but you do *not* type the @ yourself!

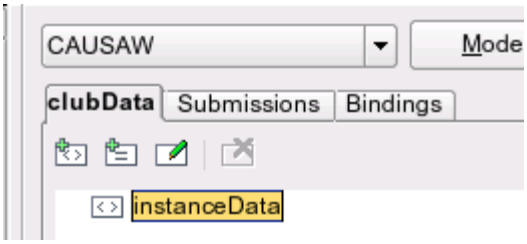

Illustration 5: Tools for modifying instance data

You end up with the hierarchy shown in Illustration 6 for the instance data.

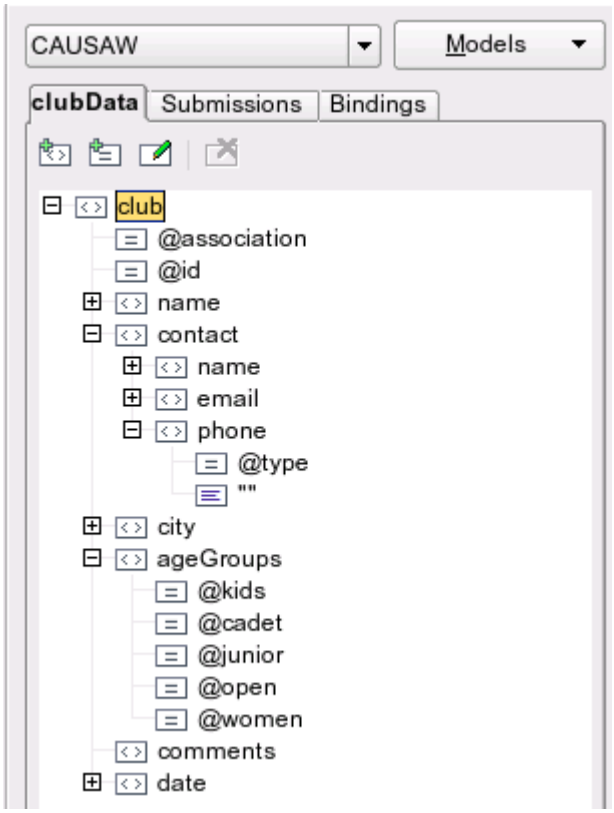

Illustration 6: Instance Data for a Club

Bindings

Let's set up the bindings for the fields that we've entered. Click the Bindings tab in the XForms dialog area. If there are already bindings there, delete them. The first binding we will add is the one for the club name.

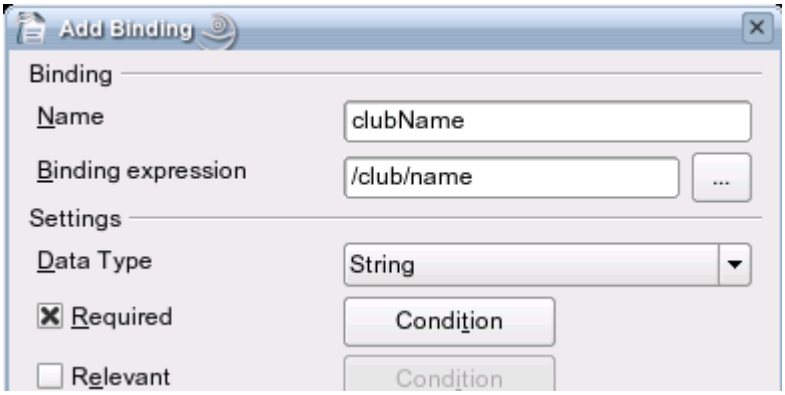

Illustration 7: Binding for Club Name

The name is up to you; the binding expression is an XPath expression that tells which part of the instance data is connected to this binding. For the purposes of this article, we can use XPath as if it were a pathname ot a UNIX file. (For those from the Windows world, the leading / is like the "top level of the drive", and directory names are separated by a slash instead of a backslash.)

Similarly, here are the other bindings to set up. You don't need to specify a data type; if you don't, any type of data is acceptable.

Binding Name	***Binding Expression***	***Required***
`contactName`	`/club/contact/name`	Yes
`phone`	`/club/contact/phone`	No
`email`	`/club/contact/email`	No
`city`	`/club/city`	Yes

We now need to connect the fields to the bindings. Double-click the text area for the club name, and click the Data tab.

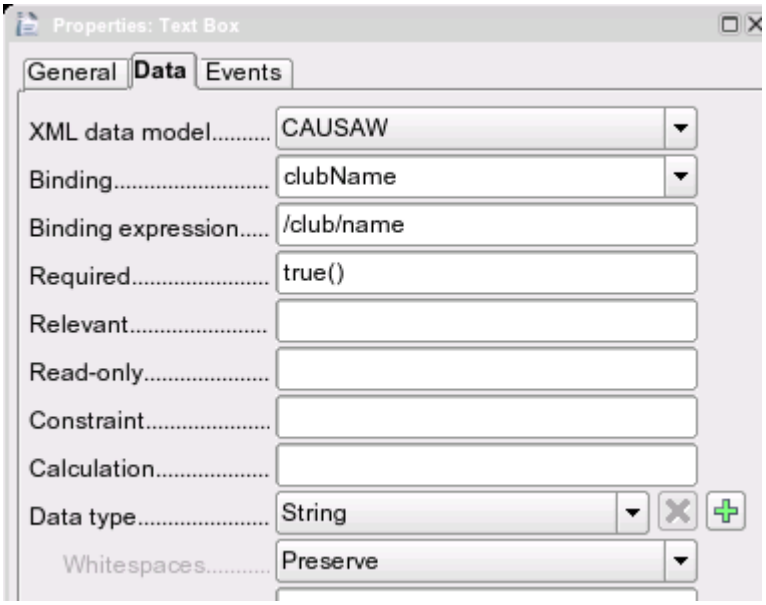

Illustration 8: Binding a form control

All you need to do is select the data model and the binding from the drop-down menus. The rest of the information will be filled in for you when you TAB out of the Binding field.

Submissions

Let's add a submission to the form to let us save the instance data in a file on our local disk. In the submissions tab, add a submission named `saveData`. Set its *action* to the path to a file. Your choices for *Method* are "put", "get", and "post". In this case, you want "put." You don't want to replace the data in the form with anything else, so set the Replace field to None.

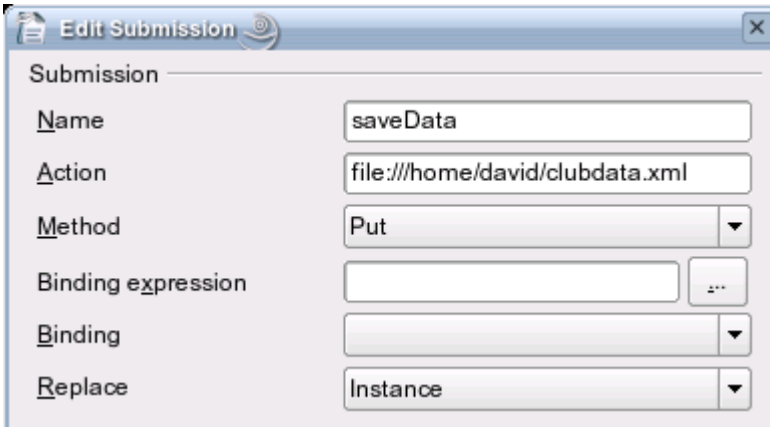

Illustration 9: Setting the data for a submission

Finally, add a button for saving the data. The label field should be set to the button legend (it's near the top of the dialog box, which is not shown here). The important thing to do is to set the button's action to submit the data, and associate it with the appropriate submission.

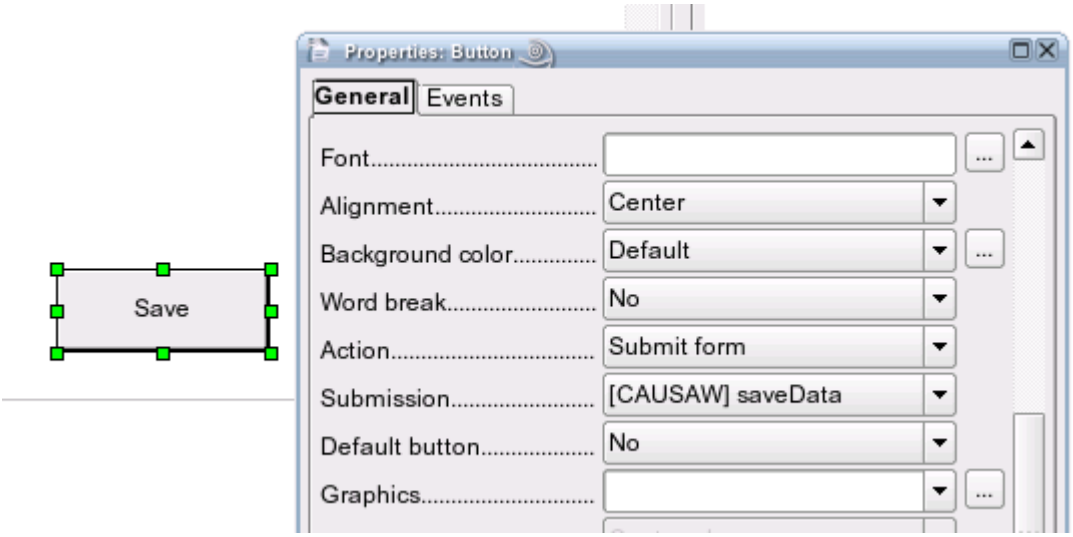

Illustration 10: Save button and its properties

Give it a try; go out of design view and fill in the form, You'll see that required fields are highlighted in red, and if you hover the mouse over a field, it will tell you so. Fill in the data and click the Save button. Now look for the `clubdata.xml` file at the path you specified, and there's your data. It will all be on one long word-wrapped line; we've split it and indented it here for ease of reading:

```
<?xml version="1.0"?>
<club association="" id="">
   <name>San Jose W-Squad</name>
   <contact>
      <name>Moishe Pipik</name>
      <email>moishe_pipik@example.com</email>
      <phone type="">408-555-7871</phone>
   </contact>
   <city>San Jose, CA</city>
   <ageGroups kids="" cadet="" junior="" open="" women=""/>
   <comments/>
   <date/>
</club>
```

Congratulations! Now that you have the basics, let's add the radio buttons and checkboxes.

Radio Buttons

For the phone number, we'll add radio buttons to select home, office, or cell. When you place the radio buttons for the type of phone, though, OpenOffice.org creates buttons labelled "Option Button". Up until now, we have changed the labels for text areas just for the sake of completeness. In this case, you *must* double-click the button to bring up the dialog box that lets you change the button's text.

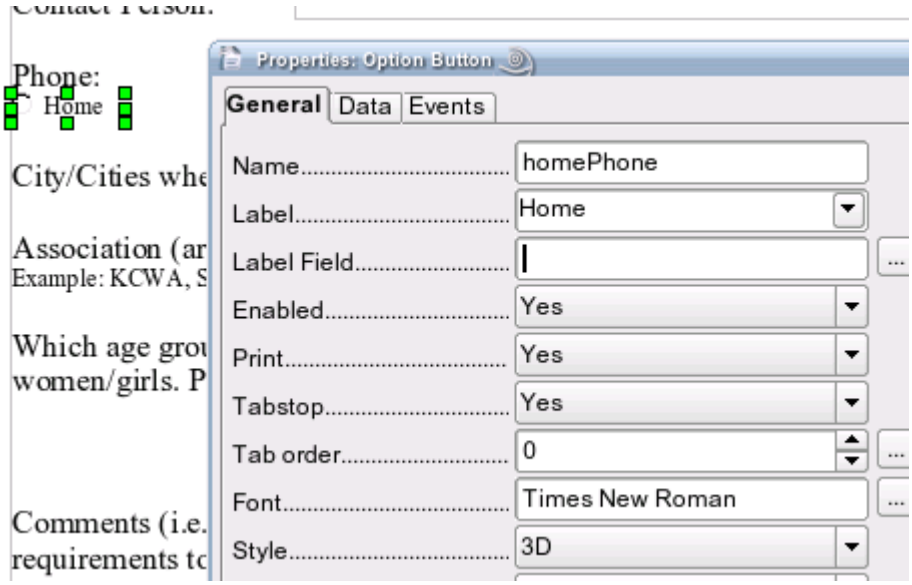

Illustration 11: Changing the Label for the radio button

Additionally, we have set the font to Times New Roman, and given the button an inset effect by setting its style to 3D. This is *not* part of XForms; it's up to an application to set the look and feel of form controls. XForms is concerned with linking together the controls with the instance data. Speaking of which, you need to set the binding for the radio buttons, which we will call phoneType. It has a binding expression of `/club/contact/phone/@type`. In order to "connect" a group of radio buttons so that only one can be chosen at a time, set all of them to have the same binding expression. Then fill in the Reference value (on) to the value you want the element to have when the button is selected. Illustration 12 shows the binding for the "Home" button.

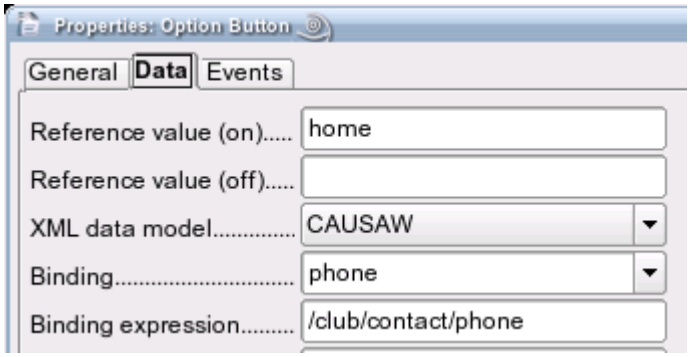

Illustration 12: Binding for home phone radio button

Checkboxes

First, add the bindings for the checkboxes, none of which is a required field.

Binding Name	***Binding Expression***
`kids`	`/club/ageGroups/@kids`
`cadet`	`/club/ageGroups/@cadet`
`junior`	`/club/ageGroups/@junior`
`open`	`/club/ageGroups/@open`
`women`	`/club/ageGroups/@women`

Now set up the bindings for the checkboxes. The unchecked boxes should not have an empty string as their value, so you will have to add a reference value for the checkbox on and checkbox off. Illustration 13 shows the dialog for the cadet checkbox:

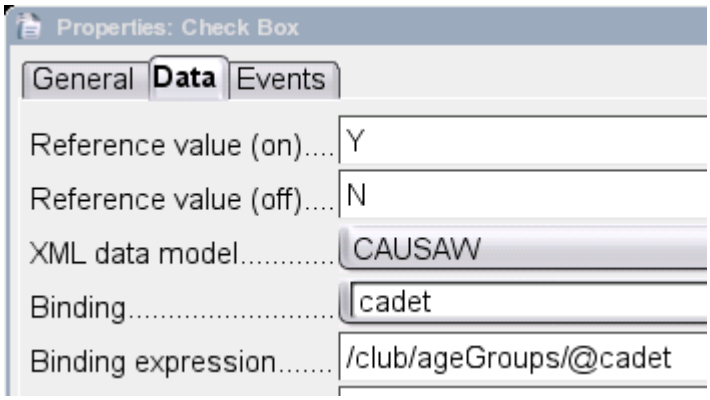

Illustration 13: Checkbox with "on" and "off" values

Drop-down Lists

The entry for the association (area) that a club is affiliated with needs special attention. The state organization for amateur wrestling has divided California into geographical areas. A club can be part of the Kern County Wrestling Association (KCWA), Santa Clara Valley Wrestling Association (SCVWA), etc. This is a crucial piece of information if you want to do an accurate search for clubs, but with the paper form, people rarely fill in this line correctly. They either leave it blank, or they put in the name of the city or school where the club meets. To solve this problem, we're going to create a drop-down list of the association names so that our club directors will always make a valid entry. If they don't know which association they belong to, that's a problem that not even XForms can solve.

It's easy enough to set the binding on the drop-down menu to `/club/@association` to determine where the chosen association will go in the instance data. But where do the list items come from? The answer is: another data model which contains that information. Create a new data model and name it `association` (click the "Models" button and select "Add"). Change the `<instanceData>` element to `<associationList>`, and then start populating it with `<association>` elements, each of which has an `abbrev` attribute. The list starts like this:

```
<associationList>
     <association abbrev="BAWA">Bay Area</association>
     <association abbrev="CAGWA">California Age Group</association>
     <association abbrev="CMWA">Coastal Mountain</association>
     <!-- etc. -->
</associationList>
```

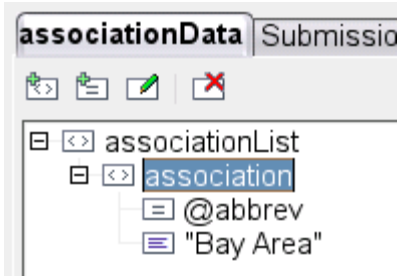

Illustration 14: Beginning of the Association Data Model

Note: the text content of the `<association>` element is displayed in the "tree," but not the attribute values.

To set the text content of the element and the attribute value, set the Default Value in the dialog. Illustration 15 shows how we entered the abbrev attribute for BAWA, the Bay Area Wrestling Association.

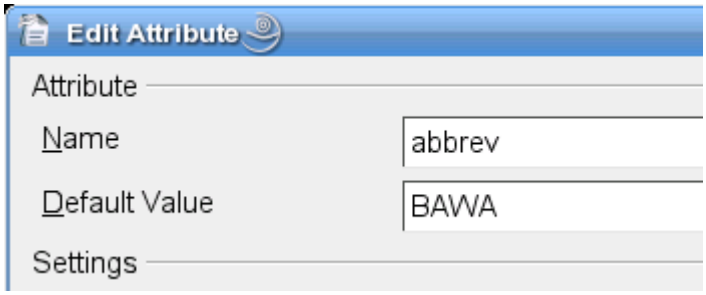

Illustration 15: Entering the Default Value for an Attribute

Truth in advertising: entering all the elements and attributes by hand takes a long while. I cheated by unzipping the file, editing the `content.xml` file, and re-zipping the file.

Illustration 16: Binding the Association drop-down menu

Now create a binding named `associationMenu` (or whatever you like,) and make the binding expression `/associationList/association/@abbrev` as shown in Illustration 16.

Finally, double-click the drop-down menu in the form and set its List Entry source to the binding you just created, as shown in Illustration 17.

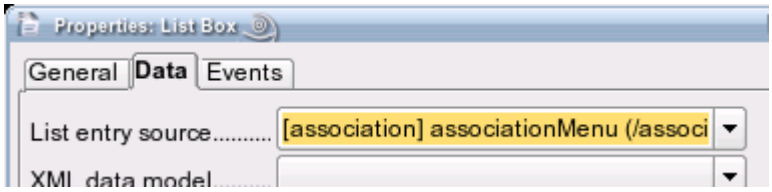

Illustration 17: Specifying the source for a list

Finishing Touches

Now, go back to the CAUSAW data model and create a binding named `comments` that has the binding expression `/club/comments`. Create a large text box in the form for the comment area, and connect it to that new binding. In the General tab, you will also want to make sure that the field is set to multi-line input, as shown in Illustration 18 (you need to scroll down in the dialog box to see this option).

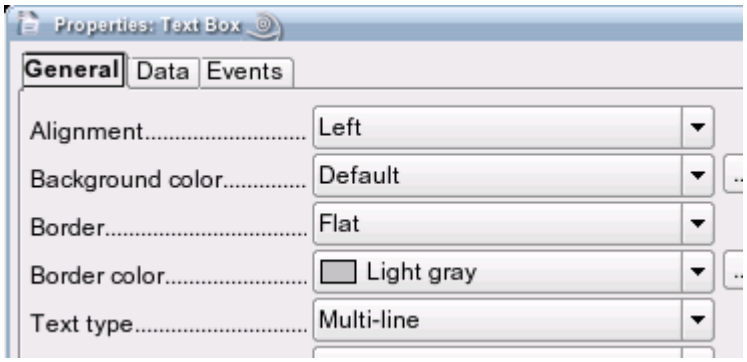

Illustration 18: Multi-line option for Text Boxes

The signature area and the admonition to "**Please try to be accurate and WRITE LEGIBLY**" are not needed, so they disappear from the form. That leaves only the date on which the form was submitted. Create a binding named `date` in the CAUSAW data model with a binding expression of `/club/date`. Make sure that you set its datatype to Date, as shown in Illustration 19. Then create a text box for the date and connect it to that binding. You may also set the data type there.

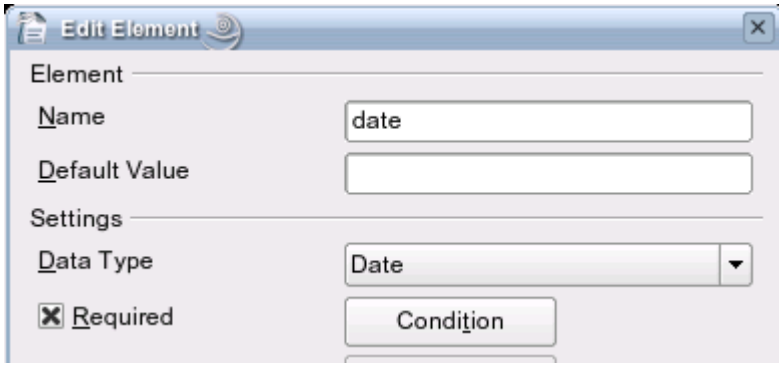

Illustration 19: Setting a Data Type

In a perfect world, OpenOffice.org would validate your input to make sure it was in proper format for a date; version 2.0.3 does not do this, though it will correctly validate a Decimal field.

Sending Form Data to a Server

To complete the form, we need to set the Club Code, which is normally assigned by the state organization. In this case, we'll create another button that, rather than save the XML in a file, will send it to a server. The server will check the ID field, and if it's not set, will assign a value. When you create this field, you set its Enabled status to No and its Read Only status to Yes. This prevents users from even trying to type in the field. The binding for this text box will be `/club/@id`.

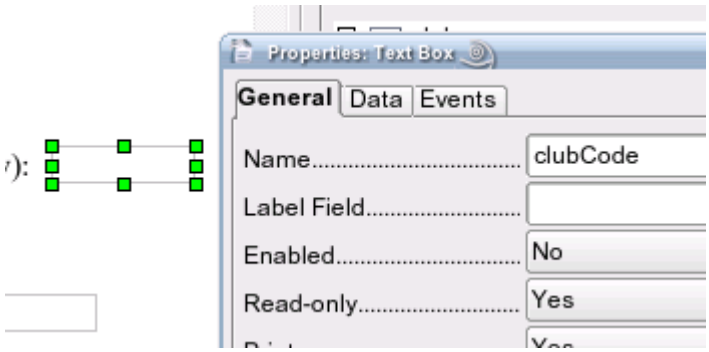

Illustration 20: Setup for the Club Code field

To send the form data, create another submission, as shown in Illustration 21. In this example, the action URL is `http://example.com/cgi-bin/process_club.cgi`, but you will change it for your server. We will use the Post method, which makes the XML data available to your CGI script's standard input. The Get method sends sets of field-name-and-value data pairs from the XML to your script, but the version of OpenOffice.org as of this writing does not handle URL-encoding properly.

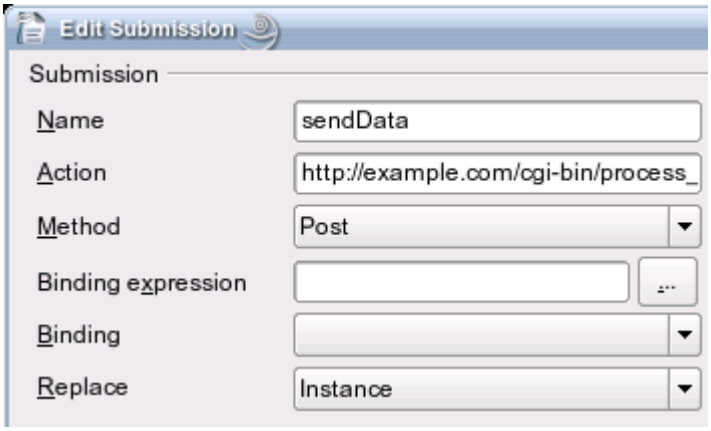

Illustration 21: Information for "Send Data" Submission

The most important item here is the Replace field—the script is going to replace the instance data that was sent to it. This means that your script *must* send back all the data that it received. If the script sends back incomplete data, the fields in the form will be blanked out. If the script sends back completely wrong data, fields in the form will give "incorrect binding" messages. This means that you should make a copy of your document, and test the script with that copy—not with the original!

Create a button with the legend *Send*, and connect it to the submission you created in the previous step. Illustration 22 shows the button and the relevant part of the properties dialog box.

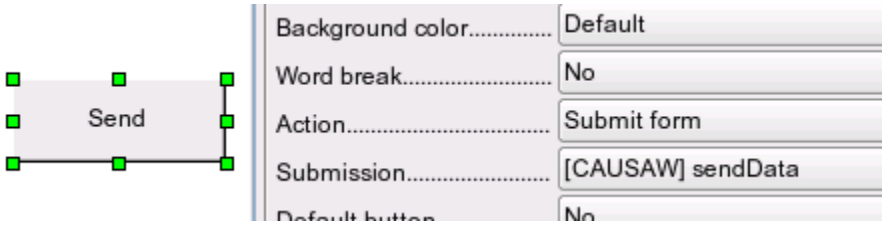

Illustration 22: Send Button with Action and Submission Properties

Appendix B shows a Perl script that parses the data, sets the ID attribute (if it's not already set), and sends the data back to the form via standard output. It's been written to show the code that's absolutely necessary, so it does very little error checking.

Appendix D shows you the completed form. To manipulate the form, you'll have to click the "Design Mode On/Off" icon in the Form Design toolbar. To see the form construction area shown in Illustration 2, click the small corrugated "Show" bar on the right side of the page, which you see magnified in Illustration 23, or click the "Data Navigator" icon in the Form Design toolbar.

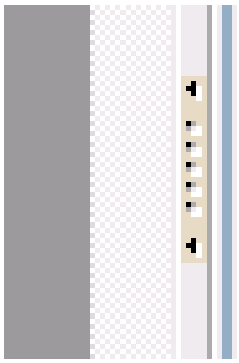

Illustration 23: "Show" bar

Extracting Data Directly From the Form

If you don't have a server, you *could* ask the directors to save the XML file and email it to you. That's an extra step for them, though, and you don't want to bother them with finding the XML file (or wondering what XML is all about). It would be better to ask them to send the document back to you, and you extract the data directly from the document. This is quite simple on a GNU/Linux system. Since the OpenOffice.org document is a .zip file, you need only unzip the file and send the content.xml file through an XSLT transformation to grab the instance data you want. Here's the command line:

```
unzip -p openofficefile content.xml | \
     xsltproc –stringparam instance instancename \
     extract_instance.xsl - > datafile
```

Where openofficefile is the name of the document containing the form, *instancename* is the id of the instance you wish to extract, and outputfile is the file that contains the XML of the instance data. You can see the XSLT file in Appendix C.

Summary

This tutorial has given you the basic concepts of XForms. You can create form controls whose data is bound to parts of an XML data instance, and you can save that data in a local file or send it to a server. While we have not explored the full capabilities of XForms, you now have enough information to start using this powerful new technology, and, we hope, explore it further.

Appendix A: Sample Club Listing Application

Club Leaders:
We have numerous inquiries as to clubs in specific areas for members to join. We can now list your **CHARTERED** club on the California USA Wrestling web page (www.ca-usaw.org), if you wish. If you authorize this advertising, we can list the information below on the web page.

CLUB NAME: _____ Club code (state use only): _____

Contact Person: _____

Phone: _____ Email: _____
Designate home or office phone

City/Cities where practices are held: _____

Association (area) affiliated with: _____
Example: KCWA, SCVWA, etc.

Which age groups do you serve? (check all appropriate division) Some clubs make a special effort to encourage women/girls. Please check if this applies to your club.

❑ Kids ❑ Cadets ❑ Juniors ❑ Open ❑ Women/Girls

Comments (i.e., practice season; cost of joining club, for example, price includes t-shirts/singlets; special requirements to join, etc.)

_____

_____

_____

_____

_____

Please complete this form and return as soon as possible if you authorize California USA Wrestling to include your club on the web page with the information you provide above.

Signature: _____ Date: _____

Please try to be accurate and WRITE LEGIBLY, as it is very time-consuming to make edits.

Appendix B: Perl Script for Handling Received Data

```
#!/usr/bin/perl
use strict;
use XML::DOM;

#  Always ensure some sort of output
#
print "Content-type: text/xml\n\n";

my $data;           # data from the form
my $parser;         # the XML parser
my $doc;            # document resulting from the parse
my $club_element;   # the <club> element node
my $club_id;        # the id attribute's value

#  Reading from standard input will set
#  $data to the instance XML.
#
read STDIN, $data, $ENV{"CONTENT_LENGTH"};

#
#  Parse the data and find the current id
#
$parser = XML::DOM::Parser->new( );
$doc = $parser->parse($data);
$club_element = $doc->getElementsByTagName("club")->item(0);
$club_id = $club_element->getAttribute("id");

#  If there's no club id, assign one. In a real
#  application, you would use a database to set the club
#  code. In this case, we just set it to "S99".
#  If the club already has an id, leave it unchanged.
#
if ($club_id eq "")
{
   $club_element->setAttribute("id", "S99");
}

#  Convert result to a string, and print it to standard output,
#  which in this case is the client XForms document.
#
print $doc->toString;
```

Appendix C: XSLT file for extracting instance data

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
    xmlns:xforms="http://www.w3.org/2002/xforms">
<xsl:param name="instance" />
<xsl:output omit-xml-declaration="yes" indent="yes" />

<xsl:template match="/">
     <xsl:if test="$instance=''">
          <xsl:message terminate="yes">
               No instance specified.
          </xsl:message>
     </xsl:if>
     <xsl:apply-templates
          select="office:document-content/office:body/office:text/
          office:forms/xforms:model/xforms:instance[@id=$instance]"/>
</xsl:template>

<xsl:template match="xforms:instance">
     <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*">
     <xsl:copy>
          <xsl:copy-of select="@*"/>
          <xsl:apply-templates/>
     </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Appendix D: The Final XForms Version of the Club Listing Form

Club Leaders:
We have numerous inquiries as to clubs in specific areas for members to join. We can now list your **CHARTERED** club on the California USA Wrestling web page (www.ca-usaw.org), if you wish. If you authorize this advertising, we can list the information below on the web page.

CLUB NAME: San Jose W-Squad Club Code (state use only): S99

Contact Person: Joe Schlabotnik

Phone: 408-555-7871 Email: joe_schlabotnik@example.com

○ Home ◉ Office ○ Cell

City/Cities where practices are held: San Jose, CA

Association (area) affiliated with:

Which age groups do you serve? (check all appropriate division) Some clubs make a special effort to encourage women/girls. Please check if this applies to your club.

☐ Kids ☒ Cadets ☒ Juniors ☒ Open ☐ Women

Comments (i.e., practice season; cost of joining club, for example, price includes t-shirts/singlets; special requirements to join, etc.)

Please complete this form and return as soon as possible if you authorize California USA Wrestling to include your club on the web page with the information you provide above.

Save Send

Date: 2006-08-14