Use XForms to create an accounting tool, Part 3: Developing asset management functionality

Building budgeting and billing forms into X-Trapolate

Skill Level: Intermediate

Stony Yakovac (syakovac@gmail.com)
Software engineer
#####

03 Apr 2007

This six-part series demonstrates how to leverage the power of XForms in conjunction with MySQL and PHP for support processing to create an online accounting tool called X-Trapolate. Every good programming technology possesses a range of problems it excels at solving. The series highlights some of the problems that the XForms solves effectively, such as the need for live calculations and greater interactivity. Part 3 of this six-part series demonstrates how to leverage the power of XForms in conjunction with PHP and MySQL to create some tools for interacting with the data of day-to-day business. Budgeting and billing forms, which we'll build into X-Trapolate in this tutorial, demonstrate some powerful features of XForms, while also highlighting the boundary where XForms development becomes difficult.

Section 1. Before you start

This tutorial is for Web application developers investigating the XForms Web form processing technology. This tutorial is the third installment of a six-part series developing a suite of accounting tools for a business. The last tutorial introduced some basic XForms flavors of classic HTML forms concepts. This tutorial introduces several more XForms programming elements, the mechanisms of which perform some fairly involved calculations.

© Copyright IBM Corporation 2007. All rights reserved.

About this tutorial

Nearly every profitable business implements some notion of billing management. The example business demonstrating the functionality for this tutorial (we will create a budgeting tool for X-Trapolate) operates under the assumption that customers make orders that consist of separate items and that each item comes from a specific department. Each order comes from a single invoice and the income from each item attributed to a department directly contributes to that department's profit. Each department also makes purchases of equipment that contributes to the expenses for that department. The budgeting process allows management to create and destroy departments and sub-departments and to change each department's planned profit and planned expenses or to compare the planned profits and expenses against the current totals as determined by the recorded orders and expenses.

The billing tool, that will also be created in this tutorial, attempts to serve as a mechanism for the billing staff to generate bills for customer accounts, print receipts, or send accounts to collections. The database of contacts originally referenced in the last tutorial of this series, after enhancements, contains information adequate for billing and can be queried for that information. Additionally, identification numbers link the accounts with individual orders and with payments.

About this series

The purpose of the series is to demonstrate the use of XForms in the development of realistic Web applications and to instruct the reader in the use of XForms.

- Part 1 is an introduction to the entire series summarizing all the portions of the end result and what facets of the XForms specification each part covers.
- Part 2 covers logging in and account management.
- Part 3 covers the development of forms pertaining to asset management.
- Part 4 continues the coverage of the development of asset management and reporting of various accounting aspects of a business.
- Part 5 covers liability management and more enhancements.
- Part 6 concludes the series with a summary of the developed tools, and some suggestions for improvement and further work for the tool set.

Prerequisites

© Copyright IBM Corporation 2007. All rights reserved.

This tutorial uses a MySQL database for storage and reference. Necessary SQL commands appear throughout the article, but require a working knowledge of MySQL. PHPMyAdmin offers equivalent access to configure the MySQL database and view the entries from a menu driven graphical interface.

Though the purpose of the series is to educate the reader about the use of XForms, some background is expected. There are some very good articles and introductory series concerning XForms available from developerWorks (see Resources). XForms is built on XML, and, hence, a basic understanding of XML is also assumed.

Other technologies and concepts may also be involved, but they will be to a much lesser extent and should be inconsequential to the reader's comprehension of the topic. Some software is also required:

- A browser capable of displaying XForms, such as Firefox 2.0.1.
- A Web server with PHP enabled, such as WAMP
- An SQL server, MySQL, which is part of the WAMP package in this case.

Section 2. Creating the budgeting form

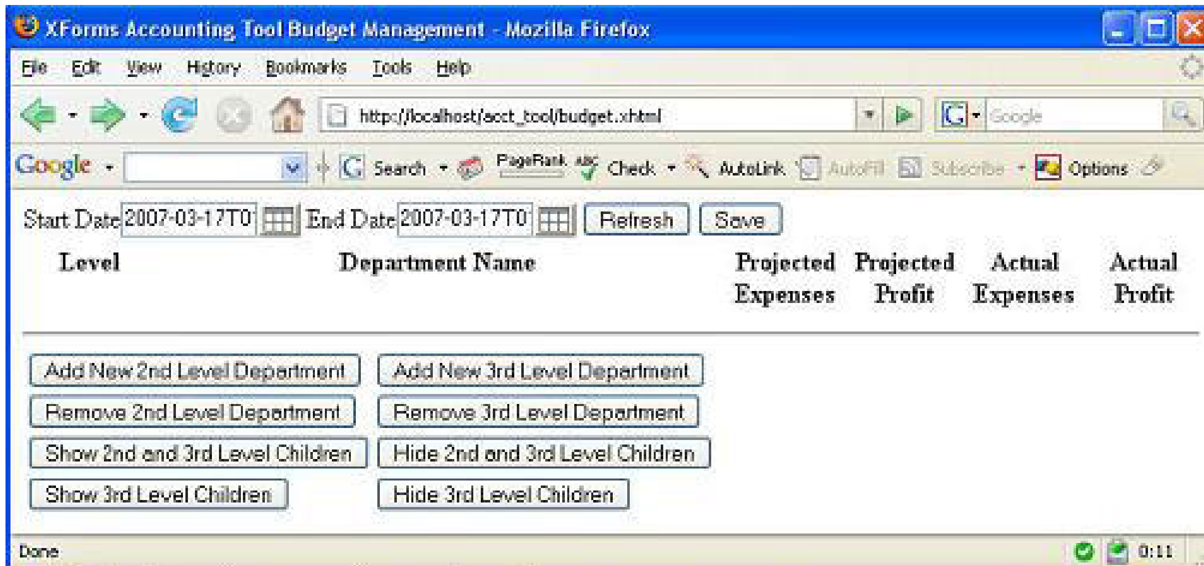
The budgeting tool intends to present a user interface that a manager can understand to develop a financial plan for the business as a long term goal. This section shows you how to create a budgeting tool using XForms with some help from MySQL and PHP.

Overview

When the user first sees the budgeting form, it appears as shown in Figure 1.

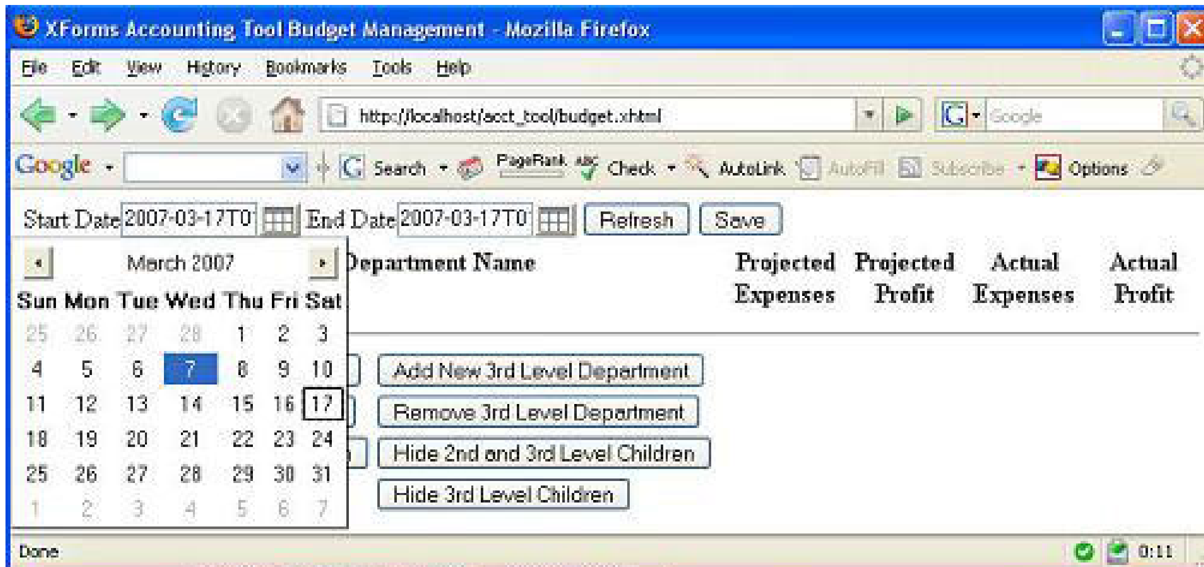
Figure 1. Budgeting form initial state

© Copyright IBM Corporation 2007. All rights reserved.



Some features of XForms that this tutorial has not previously discussed appear in this display. The date box is a new feature, which acts similar to a drop-down box, but the user can instead select a day from the calendar that pops up, as shown in Figure 2.

Figure 2. XForms calendar for date selection



Next, you'll see how to code the date datatype and calendar selection.

Date datatype and calendar selection

As mentioned before, XForms offers independence of the data model from the presentation as a key feature. The browser actually chooses a presentation that makes sense for that browser. It might be expected that the date entry definition

© Copyright IBM Corporation 2007. All rights reserved.

would contain some new and unique widget to make it appear as a date selection box. Code Listing 1 shows the code for the date boxes.

Listing 1. Date selection box definition

```
   <xforms:input
ref="instance('acctToolBdgtInst')/startDate">
<xforms:label>Start
Date</xforms:label>
</xforms:input>
```

Notice that the date selection is identical to the username definition in Part 2. The browser gets the clue to treat this box differently from another location. The mechanism for creating the date selection box appears in a bind statement in the XForms model declaration. The bind statement in question appears in Listing 2.

Listing 2. defining a date datatype

```
<xforms:bind
nodeset="instance('acctToolBdgtInst')/startDate"
type="xsd:date"/>
```

The type attribute applies to the `startDate` node in the `acctToolBdgtInst` instance using the bind statement. The declaration of the `startDate` node as an `xsd:date` allows it to appear as a calendar selection box. The browser performs the function of deciding how to display the input entry, but that type drives that decision. Very few data types change the rendering of form controls in the browser, but some data types apply. As XForms technology spreads and becomes more mature, browsers may adapt to treat more data types specially. The browser requires one other key piece of information required to make the type attribute mean anything. This point can frustrate a new user of XForms. The code listing in Listing 3 defines the name spaces used in an XHTML document. After some trial and error, this specific combination of the declaration of the `xsd` namespace and `xsd:date` allowed Firefox to create a calendar entry.

Listing 3. defining a date datatype

```
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:my="http://example.com/my"
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xhtml="http://www.w3.org/1999/xhtml"
>
```

Next, you'll initialize the form's instance data.

© Copyright IBM Corporation 2007. All rights reserved.

Initializing XForms instance data

One less obvious feature shown in Figure 2 is the fact that the date box appears with the current date in it. Though several options exist for putting values into XML nodes, most of the options don't apply well to the desired behavior of this case, which is to load the current date into the node. Part 2 demonstrated some methods such as hard coding a value into the XML declaration of the data structure. That, of course, configures that node to a fixed value at startup and, when the application starts the next day, the date is no longer current. Binding and using the `calculate` property allows a programmer to instruct the browser to set a value for the node; but, in this case, binding the node to `now()` would yield a control the user could not change, or, at best, would change whenever the browser performed a calculation, which the user of the application certainly would not understand.

Using the XForms-ready event

The node needs a value stored in it at the loading of the page and never again, unless the user sets the value. The code in Listing 4 shows the XForms code to perform that activity. Part 2 introduces an event attached to a trigger. The `DOMActivate` event appears inside the declaration of the trigger in the login form. The budget uses a different event declared in a different location. The `xforms-ready` event fires when the browser has finished loading the XForms structures, but before control transfers to the user. Another element in XForms processing, the setvalue element appears here.

Listing 4. Setting the date on the XForms-ready event

```
<xforms:action
ev:event="xforms-ready">
<xforms:setvalue
ref="instance('acctToolBdgtInst')/startDate"
value="now()"/>
<xforms:setvalue
ref="instance('acctToolBdgtInst')/endDate"
value="now()"/>
</xforms:action>
  </xforms:model>
</head>
```

The setvalue element evaluates the value attribute and applies that value to the node indicated by the ref attribute. Other incantations of setvalue exist, but this particular flavor allows the use of XPath to evaluate the `now()` function, which returns the current date and time. The following snippet of code shown in Listing 5 comes from the billing form and demonstrates the use of the XPath function `substring` to trim the `now` result to just the date.

© Copyright IBM Corporation 2007. All rights reserved.

Listing 5. Trimming now to just the date

```
<xforms:setvalue
ref="instance('acctToolBillInst')/endDate"
value="substring(now(),1,10)"/>
```

Data validity checks and XML Schema datatypes

The reason the billing form requires the modified result from `now` comes from the defined datatype for the field. The XML Schema datatype `date` serves as an XML Schema constraint for the data in that field. Firefox evaluates the legality of that data when a submit occurs. By leaving the unmodified result of `now` in the date boxes, the submit process complains about not being able to validate the data. Unfortunately, when that happens, sometimes the browser replies with a comment about "Not able to validate," but other times the browser does not reply at all, leaving the user clicking the button repeatedly and wondering what could possibly be happening.

The budget form does not specifically require the date trimming because it doesn't use the instance with the date in it for submission. This fact is shown in Listing 6. Enhancing the query to use the dates as constraints for the query require that instance data as an argument, and hence, require the XML Schema validity for the start and end dates, which are declared as `xsd:date`. The current argument for the `load_budget` submission method is the `departmentsInst` instance. The argument node, `acctToolBdgtInst`, which contains the startDate and endDate, both constrained as xsd:date, requires trimming the time off the `now()` result before the submission method is executed.

Listing 6. Avoiding the XML Schema validity check

```
<xforms:instance
id="acctToolBdgtInst">
         <formui
xmlns="">
<startDate/>
<endDate/>
      ...
</xforms:instance>
      ...
<xforms:submission
id="load_budget"
action="get_budget.php"
method="get"
replace="instance"
instance="departmentsInst"
ref="instance('departmentsInst')"
/>
      ...
```

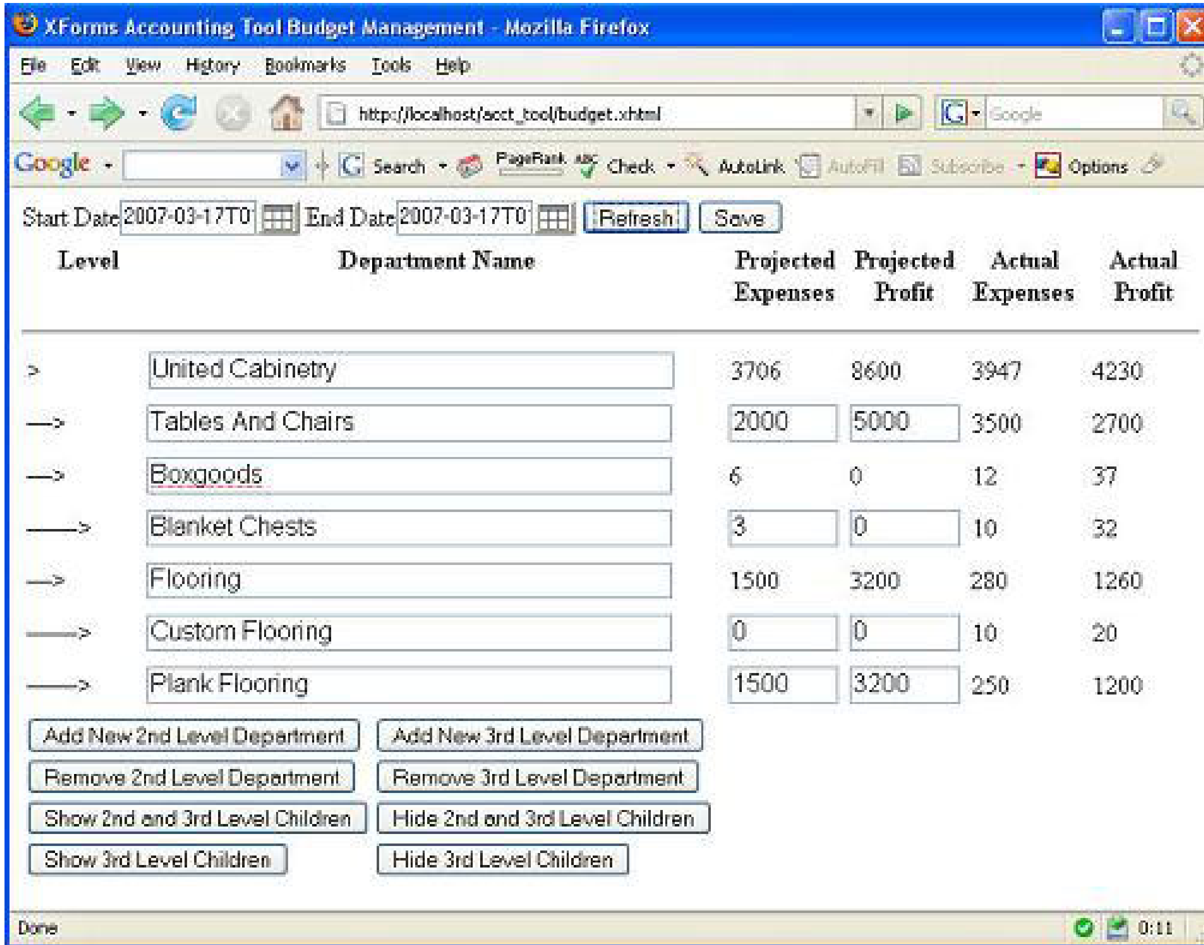
Next, you'll get a taste of what the budget form will look like.

© Copyright IBM Corporation 2007. All rights reserved.

First look at departments

When the user activates the Refresh button, a display the same or similar to the one in Figure 3 appears. The Refresh button XForms code consists of a trigger, which activates the submission mechanism. This implementation, functionally identical to a regular submit element, demonstrates that a form can have and use a submission element with or without a submit element.

Figure 3. Budgeting after refresh



The subtle difference between the submit and trigger implementations comes from Firefox placing submit elements on a new line unless told not to, and placing trigger elements on the same line. Listing 7 shows the trigger code.

Listing 7. Triggering a submit without the submit element

```
<xforms:trigger
ref="instance('acctToolBdgtInst')/loadBudget">
<xforms:label>Refresh</xforms:label>
<xforms:action
```


© Copyright IBM Corporation 2007. All rights reserved.

```
ev:event="DOMActivate">
<xforms:dispatch
name="xforms-submit"
target="load_budget"/>
</xforms:action>
</xforms:trigger>
```

Next you'll see how you can use the action and dispatch elements in XForms.

Using action and dispatch in XForms

The budget form could have performed satisfactorily with an XForms submit element, but using trigger presents an interesting option. The action element in XForms executes the elements it contains sequentially. Using this mechanism, a single button can drive multiple submission mechanisms or perform other operations. The dispatch element serves as the driver of an event. By using dispatch, nearly any action an XForms form can perform becomes available. This instance, shown in Listing 7, used the `xforms-submit` event which, when targeted at the `load_budget` target, causes the `load_budget` submission method to execute. The load_budget submission element is shown in Listing 6.

Next, you'll take a look at some the XML instance data for the budgets form.

The budget form's XML data structure

At this point, it is necessary to introduce the data structure for the departments. This example uses a hierarchical data structure of departments in the XML. Listing 8 shows an abbreviated example of how the XML is structured. This hierarchical structure allows the calculations to pertain to each department and its subdepartments. An alternative implementation sets the XML nodes all at the same level but adds informative nodes to create the concept of the hierarchy. The calculations become an insurmountable difficulty for that method.

Listing 8. Hierarchical department XML

```
<?xml
version="1.0"
encoding="UTF-8"?>
<budgetForm
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events">
  <departments>
    <maxid/>
<hidden>0</hidden>
    <department
id="9">
<name>United
Cabinetry And
Millwork</name>
```


© Copyright IBM Corporation 2007. All rights reserved.

```
      <department
id="3">
<name>Boxgoods</name>
<department
id="4">
<name>Blanket
Chests</name>
```

The hierarchical method presents some interesting difficulties. The main difficulty comes from what can almost be done. The code shown in Listing 10 creates the display in Figure 3. Listing 9 presents a much simpler repeat version, that, in the end does not work because the repeat element considers the nodelist as a flat, parallel structure that can be selected with an integral index for operations such as insert. The Listing 9 code uses the XPath `//` expression to hierarchically search all nodes under departments named department and use them as the nodeset. Ultimately, Listing 10 provides a working model where each repeat nodeset uses all the department nodes at a single level and none below. This method allows the insertion of new nodes into the department's tree.

Listing 9. Hierarchical dynamic depth department display

```
   <xforms:repeat
id="budgetRpt"
nodeset="instance('departmentsInst')/departments//department"
incremental="true">
```

Nesting repeat elements

Listing 10 shows how to use nesting repeat structures in XForms. This example uses a three-level nesting. Each repeat has many other XForms elements and contains an ID attribute. The ID attribute provides access to the index of the specific repeat structure. The XForms repeat behaves much like any of the other more direct user interaction-based form controls. An important behavior to understand about the repeat is that the last line the user clicked on is remembered for that repeat and is available using a function. Later this index will be used to show the insert functionality.

Listing 10. Hard-coded depth department display

```
   <xforms:repeat
id="budgetRpt"
nodeset="instance('departmentsInst')/departments/department"
incremental="true">
     <table
width="750px">
       <tr>
         <td
width="10%">
           >
         </td>
```


© Copyright IBM Corporation 2007. All rights reserved.

```
         <td
width="50%">
<xforms:textarea
ref="name"/>
         </td>
         <td
width="10%">
<xforms:textarea
ref="planExp"/>
<xforms:output
ref="planExpDisplay"/>
         </td>
         <td
width="10%">
<xforms:textarea
ref="planPro"/>
<xforms:output
ref="planProDisplay"/>
         </td>
         <td
width="10%">
<xforms:output
ref="curExp"/>
         </td>
         <td
width="10%">
<xforms:output
ref="curPro"/>
         </td>
       </tr>
     </table>

<xforms:repeat
id="budgetRpt2"
nodeset="department"
incremental="true">
       <table
width="750px">
         <tr>
           <td
width="10%">
             ->
           </td>
           <td
width="50%">
<xforms:textarea
ref="name"/>
           </td>
           <td
width="10%">
<xforms:textarea
ref="planExp"/>
<xforms:output
ref="planExpDisplay"/>
           </td>
           <td
width="10%">
<xforms:textarea
ref="planPro"/>
<xforms:output
ref="planProDisplay"/>
           </td>
           <td
width="10%">
<xforms:output
ref="curExp"/>
           </td>
           <td
width="10%">
```


© Copyright IBM Corporation 2007. All rights reserved.

```
<xforms:output
ref="curPro"/>
           </td>
         </tr>
       </table>

<xforms:repeat
id="budgetRpt3"
nodeset="department"
incremental="true">
         <table
width="750px">
           <tr>
             <td
width="10%">
—>
</td>
             <td
width="50%">
<xforms:textarea
ref="name"/>
</td>
             <td
width="10%">
<xforms:textarea
ref="planExp"/>
<xforms:output
ref="planExpDisplay"/>
</td>
             <td
width="10%">
<xforms:textarea
ref="planPro"/>
<xforms:output
ref="planProDisplay"/>
</td>
             <td
width="10%">
<xforms:output
ref="curExp"/>
</td>
             <td
width="10%">
<xforms:output
ref="curPro"/>
</td>
           </tr>
         </table>

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

Another important feature of a repeat structure is the re-definition of the current context for XPath queries. The second two-repeat structures use the nodeset department. Though they reference the same text, they each select a different set of nodes. The reason for this is the incremental re-definition of the current location context. Since the XPath expressions do not contain a call to `instance()` or a leading `/` or some other modifier, they are referenced to the current context, which is one of the nodes selected by the XPath expression in the nodeset attribute of the repeat element.

© Copyright IBM Corporation 2007. All rights reserved.

Textarea form control

A new XForms form display control was shown in Listing 10. The textarea control allows the user to enter multiple lines of data. Clicking **Enter** in a textarea form control goes to the next line of text instead of triggering an action. For this instance, the input appears the more suitable solution. However, as Figure 4 shows, the result of replacing the textarea elements with input elements does not yield a very pretty picture. Notice that the alignment of the department names goes awry and the numbers become left justified in the input and right justified as values.

Figure 4. Using input instead of textarea

Level	Department Name	Projected Expenses	Projected Profit	Actual Expenses	Actual Profit
>	United Cabinetry	48500	93200	3947	4230
-->	Tables And Chairs	2000	5000	3500	2700
-->	Boxgoods	15000	35000	12	37
--->	Blanket Chests	5000	15000	10	32
--->	Dressers	10000	20000	0	0
-->	Flooring	31500	53200	280	1260
--->	Custom Flooring	30000	50000	10	20

Also, keep in mind that the textarea is controlled using the following CSS, as shown in Listing 11.

Listing 11. Controlling the appearance of textarea input boxes

```
    <style
type="text/css">
      textarea {
font-family:
sans-serif;
height: 1.2em;
width: 90%; }
    </style>
    <xforms:model
id="acctToolBillModel"
>
...
```


© Copyright IBM Corporation 2007. All rights reserved.

The above CSS makes the textarea boxes line up nicely, as you've already seen in Figure 3.

Next, you'll see how to insert new department elements.

Inserting new departments

The insert functions on the budget form operate as shown in Figure 5. The insert element performs the operation of creating a data node in the XML data structure. XForms re-processes the data structure and updates the display to reflect the new data node. The insert element requires two nodes, a nodeset, and an index to insert a new element. The caveat of insert comes with the fact that the new node that insert creates is created by cloning the last node in the nodeset. Two potential difficulties arise from this behavior. The first difficulty comes from the data that may or may not reside in the node being cloned. In the budgeting case, a second-level department may or may not have a third-level department below it.

Figure 5. Inserting a department

© Copyright IBM Corporation 2007. All rights reserved.

XForms Accounting Tool Budget Management - Mozilla Firefox

Start Date 2007-03-17T0 End Date 2007-03-17T0 Refresh Save

Level	Department Name	Projected Expenses	Projected Profit	Actual Expenses	Actual Profit
>	United Cabinetry	45203	48800	4197	5430
-->	Tables And Chairs	2000	5000	3500	2700
-->	Boxgoods	40003	37000	12	37
---->	Blanket Chests	10000	15000	10	32
---->	Dressers	30000	22000	0	0
-->	Enter new department name	1500	3200	250	1200
---->	Enter new department name	0	0	0	0
---->	Plank Flooring	1500	3200	250	1200
-->	Flooring	1500	3200	280	1260
---->	Custom Flooring	0	0	10	20
---->	Plank Flooring	1500	3200	250	1200

Add New 2nd Level Department | Add New 3rd Level Department
Remove 2nd Level Department | Remove 3rd Level Department
Show 2nd and 3rd Level Children | Hide 2nd and 3rd Level Children
Show 3rd Level Children | Hide 3rd Level Children

The second difficulty appears as a corollary to the first problem. If there were no nodes to start with, nothing can be created because there is nothing to clone. These behaviors create some interesting challenges in developing XForms applications with XForms 1.0, but, in this case, if the user wants a third-level department, the answer is to insert another second-level department, which will have a third level attached to it for free, compliments of a dummy node used for this specific purpose. The code for a third-level insert is shown in Listing 12.

Listing 12. Third-level insert

```
<xforms:trigger
ref="instance('acctToolBdgtInst')/newDept3rd">
<xforms:label>Add
New 3rd Level
```


© Copyright IBM Corporation 2007. All rights reserved.

```
Department</xforms:label>
  <xforms:action
ev:event="DOMActivate">
<xforms:insert
nodeset="instance('departmentsInst')/departments/department
[index('budgetRpt')]/department[index('budgetRpt2')]/department"
at="index('budgetRpt3')"
position="after"/>
<xforms:setvalue
ref="instance('departmentsInst')/departments/department[index
('budgetRpt')]
/department[index('budgetRpt2')]/
department[index('budgetRpt3')+1]/@id"
value="instance('departmentsInst')//departments/maxid"/>
<xforms:setvalue
ref="instance('departmentsInst')/departments/department

[index('budgetRpt')]/department[index('budgetRpt2')]/department[index
('budgetRpt3')+1]/name"
value="'Enter new
department
name'"/>
<xforms:setvalue
ref="instance('departmentsInst')/departments/department
[index('budgetRpt')]/department[index('budgetRpt2')]/department[index
('budgetRpt3')+1]/planPro"
value="0"/>
<xforms:setvalue
ref="instance('departmentsInst')/departments/department
[index('budgetRpt')]/department[index('budgetRpt2')]/department[index
('budgetRpt3')+1]/planExp"
value="0"/>
<xforms:setvalue
ref="instance('departmentsInst')/departments/department
[index('budgetRpt')]/department[index('budgetRpt2')]/department[index
('budgetRpt3')+1]/ordersTotal"
value="0"/>
<xforms:setvalue
ref="instance('departmentsInst')/departments/department
[index('budgetRpt')]/department[index('budgetRpt2')]/department[index
('budgetRpt3')+1]/expensesTotal"
value="0"/>
</xforms:action>
</xforms:trigger>
```

XPath filters, repeat indices, and setvalue

Listing 12 contains a lot of code; however, closer inspection reveals that most of the code performs identical functions, leaving two items to discuss, the insert and the many setvalues. The insert element does as the last section suggested; it copies a node and puts the new copy in the data structure. Insert requires a nodeset, and a location or index. The nodeset expression, an XPath expression, covers the most whitespace in Listing 12, looking like a magical monster. However, the expression simply serves to index down to the third-level node using the indices from the repeat elements discussed earlier. A note should be made that the `[]` operator in an XPath expression does not do what the typical `[]` operator of most languages does. In XPath, the `[]` operator serves as a filter. In this case, a single number is placed in the filter that selects that index; however, many things can reside in the filter.

© Copyright IBM Corporation 2007. All rights reserved.

The setvalue lines serve to initialize the data. As mentioned earlier, since the data is a copy, it may have bits of information that either don't apply, or can even cause problems. Here, one of the difficulties is maintaining unique department IDs. That is managed using the maxId value in the departments top-level node. It in turn is bound with an XPath function to be the maxId (plus one) in the entire nodeset.

Next, you'll implement code to remove existing second-level nodes.

Removing nodes

Listing 13 shows the code for the remove node buttons. Much simpler than the insert feature, the remove node references the repeat index. The delete element deletes the XML data structure referenced and, hence, the displayed data associated with that element is deleted.

Listing 13. Removing second-level nodes

```
<xforms:trigger
ref="instance
('acctToolBdgtInst')/remove">
<xforms:label>Remove
2nd Level
Department</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:delete
nodeset="instance
('departmentsInst')/departments/department[index
('budgetRpt')]/department"
at="index('budgetRpt2')"/>
</xforms:action>
</xforms:trigger>
```

Hiding and showing nodes

The budgeting form uses the relevant property to hide and show departments. This feature does not function very well in this case because of the indent indicating the level of the department. A setvalue element performs this function much like the initializing function in the insert procedure. Because of the relevant property bound to all the display nodes, hiding nodes becomes a matter of setting the hidden nodes to a value of 1 and vice versa for show, as shown in Listing 14.

Listing 14. Hiding and showing nodes

```
<xforms:trigger
ref="instance('acctToolBdgtInst')/show2nd">
<xforms:label>Show
2nd and 3rd Level
Children</xforms:label>
```


© Copyright IBM Corporation 2007. All rights reserved.

```
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance
('departmentsInst')//departments/department[index
('budgetRpt')]/hidden"
value="0"/>
</xforms:action>
</xforms:trigger>
        </td>
        <td>
<xforms:trigger
ref="instance
('acctToolBdgtInst')/hide2nd">
<xforms:label>Hide
2nd and 3rd Level
Children</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance
('departmentsInst')//departments/department[index
('budgetRpt')]/hidden"
value="1"/>
</xforms:action>
</xforms:trigger>
```

Next, you'll see how to work a little magic with the department leaf nodes.

How departments with sub-department's budgets are read-only

For budgeting purposes, a department's budget is determined by the sum of the budgets of the departments underneath it. The visual clue to the read-only nature of a super-department is that the number is no longer alterable. Each of the budgeting values, profit, and expenses, has two form controls. By enabling the textarea form control, the value becomes read-write and the visual clue that the department is a leaf department is made. Conversely, enabling the output form control creates a read-only display, which indicates the department has sub-departments. The switching between read-only and read-write and the numeric calculation behaviors happen as a result of bindings to relevant and calculate. Listing 15 shows the code for the bindings.

Listing 15. Calculations and relevant

```
<xforms:bind
nodeset="instance
('departmentsInst')//department/planPro"
relevant="../department/@id=false()"/>

<xforms:bind
nodeset="instance
('departmentsInst')//department/planExp"
relevant="../department/@id=false()"/>

<xforms:bind
nodeset="instance
```


© Copyright IBM Corporation 2007. All rights reserved.

```
('departmentsInst')//department/leafPlanExp"
relevant="../department/@id=false()"
calculate="../planExp"/>

<xforms:bind
nodeset="instance
('departmentsInst')//department/leafPlanPro"
relevant="../department/@id=false()"
calculate="../planPro"/>

<xforms:bind
nodeset="instance
('departmentsInst')//department/planProDisplay"
calculate="sum(..//leafPlanPro)"
relevant="../department/@id=true()"/>

<xforms:bind
nodeset="instance
('departmentsInst')//department/planExpDisplay"
calculate="sum(..//leafPlanExp)"
relevant="../department/@id=true()"/>

<xforms:bind
nodeset="instance
('departmentsInst')//department/curPro"
calculate="round(sum(..//ordersTotal))"/>

<xforms:bind
nodeset="instance
('departmentsInst')//department/curExp"
calculate="round(sum(..//expensesTotal))"/>

<xforms:bind
nodeset="instance
('departmentsInst')//departments/maxid"
calculate="max(//department/@id)+1"/>
```

The XPath expressions all use the `//` operator to select all department nodes. The nodes are then further dereferenced such that all the planPro fields in all the department nodes have the relevant binding placed on them. Recall from the last tutorial that the `@` character specifies an attribute. ID is an attribute, and hence, is referenced with the `@` character in the XPath expression. The comparison is to `true()` or `false()`. Those special XPath functions evaluate to Boolean true and false values. Used in the way shown in Listing 15, the comparison returns true if the node referenced exists. Also note the use of the round and sum XPath functions. See also the max function in action above when the //departments/maxid element is set. The maxid is used for setting the IDs of new rows that get added, guaranteeing distinct IDs.

Next you'll learn a little about how to populate the form's instance data using PHP's DOM.

Brief mention on budget form population using PHP

Because XForms requires the entire basic structure of the XML data, the data structure must be created initially by the PHP script. The PHP script attempts to

© Copyright IBM Corporation 2007. All rights reserved.

perform as little function as possible; however, this data also requires a hierarchical structure from an SQL database table, which is a flat structure. The get_budget PHP script performs a single task: it retrieves all the departments from the department table in the SQL database, and re-constructs the hierarchy based on the ID of each node and the stored ID of the node's parent. The value 0 is assumed to be the top node.

The SQL queries for this task contain very little of interest. The SQL database is queried using PHP, as shown in Listing 16. The algorithm used is a depth-first traversal, where a node is popped off a queue and referenced as the parent node every iteration. The ID for that node is queried as a Parent, which returns all the child nodes for that department.

Listing 16. SQL query for department

```
      $query =
sprintf('SELECT *
FROM
`departments`
WHERE Parent=%s
AND
NOT(department_id=%s);',$parentId,
$parentId);
      $queryData
=
mysql_query($query,
$sqldb);
      if
(!$queryData) die
('Could not
execute acct db
query:
<b>'.$query.'</b>
because:  ' .
mysql_error());
      while ($row
=
mysql_fetch_assoc($queryData))
{
```

The XML document initializes by a static string, as shown in Listing 17. Using that as the basis, new nodes are created and added iteratively, progressing down the tree in a depth-first fashion. The process of adding new nodes and using the linked list structure of the DomNodes in PHP is shown in Listing 17.

Listing 17. DOMNode XML tree building and DomDocument initialization

```
   $doc = new
DomDocument('1.0');
$doc->loadXml('<?xml
version="1.0"
encoding="UTF-8"?>
<budgetForm
xmlns:xforms="http://www.w3.org/2002/xforms"
xmlns:ev="http://www.w3.org/2001/xml-events">
```


© Copyright IBM Corporation 2007. All rights reserved.

```
   <departments>
     <maxid/>
<hidden>0</hidden>
   </departments>
</budgetForm>');
...
do
{
...

if($curParent->previousSibling
&& $curParent-
>previousSibling->nodeName
== 'department')
$curParent =
$curParent->previousSibling;
      else
$curParent =
$curParent->childNodes->item($curParent-
>childNodes->length-1);

      $parentId =
$curParent->getAttribute('id');

   } while
($parentId);
```

Care must be taken while processing the nodes to not truly treat the node list as a queue using the remove function, because the node truly goes away and the XML does not contain any nodes as an end result.

That's a wrap for the budget form! Note that not everything for it was covered, but you can view the complete code for the form, including supporting PHP and SQL instantiation for your database, in the provided download.

Section 3. The Billing form

The billing form uses many of the same concepts as the budgeting form with a few new additions. The billing XForms form introduces the procedure for opening a new window to display information appropriate for printing as a bill. This section implements this form, the initial state of which is shown in Figure 6.

Figure 6. Initial state of the billing form

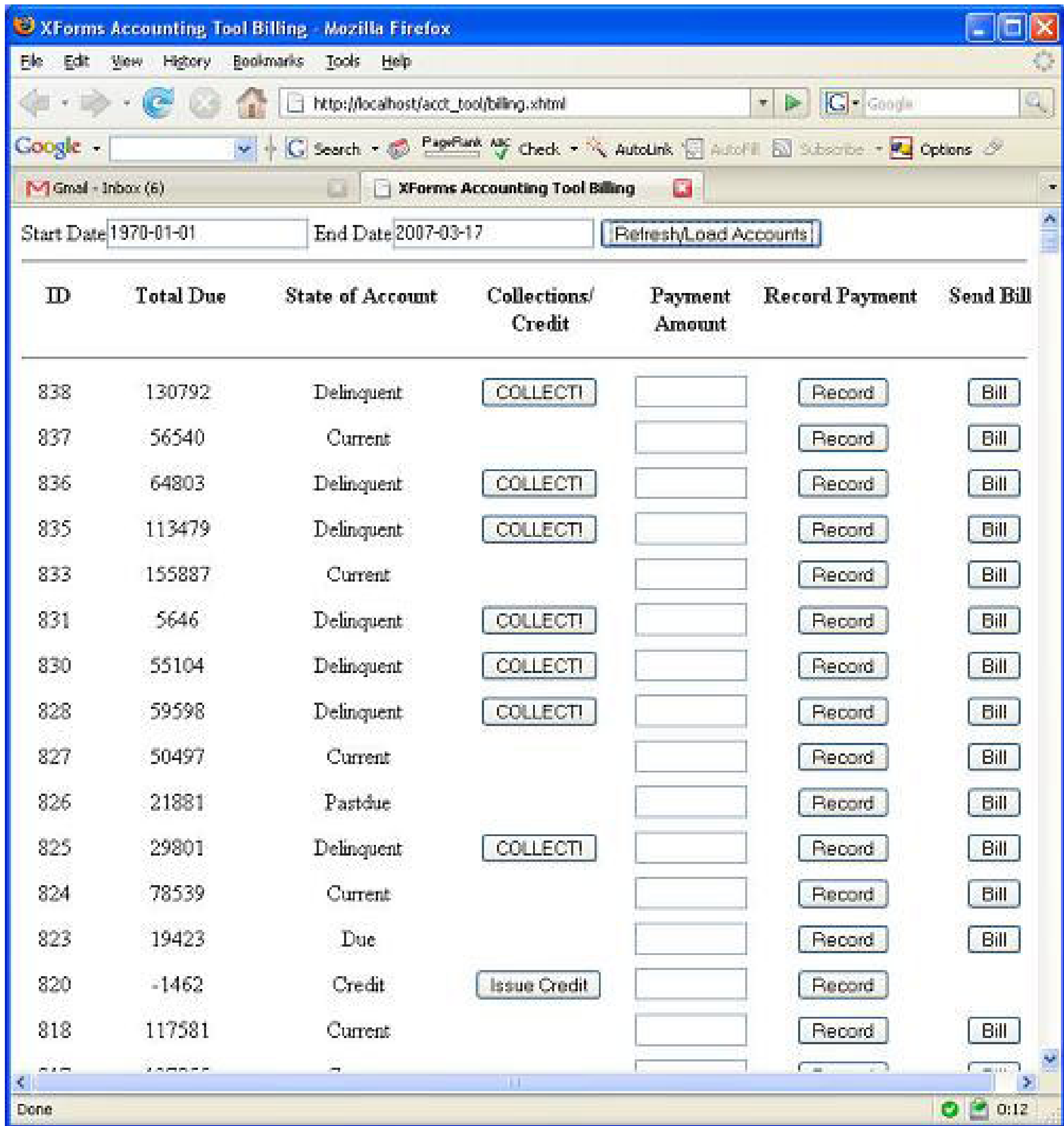
© Copyright IBM Corporation 2007. All rights reserved.

The table display

Though HTML tables do not support XForms and Firefox has not implemented the XForms repeat attribute, it is possible to repeat the entire table structure. The billing XForms form uses that mechanism to display the billing data. The display comes from a single-level repeat structure, as shown in Listing 18. Figure 7 shows the form after clicking the **Refresh/Load Accounts** button.

Figure 7. The refreshed billing form

© Copyright IBM Corporation 2007. All rights reserved.



Listing 18. Visual XForms controls for the billing form

```
<xforms:repeat
id="acctRpt"
nodeset="instance
('accts')//account">
      <table
width="750px">
        <tr>
          <td
align="center"
width="5%">
<xforms:output
```


© Copyright IBM Corporation 2007. All rights reserved.

```
ref="acctId"/>
          </td>
          <td
align="center"
width="15%">
<xforms:output
ref="creationDate"/>
          </td>
          <td
align="center"
width="15%">
<xforms:output
ref="totalDue"/>
          </td>
          <td
align="center"
width="15%">
<xforms:output
ref="accountState"/>
          </td>
          <td
align="center"
width="15%">
<xforms:trigger
style="display:
inline"
ref="submitCollection">
<xforms:label>COLLECT!</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="@makeDelURI"
value="concat('doBillingAction.php?
action=delinquent&acct=',../../acctId,'&amount=',../../payment
Amt,'&bal=',../../totalDue)"/>
<xforms:load
ref="@makeDelURI"
show="new"/>
</xforms:action>
</xforms:trigger>
</xforms:trigger>
<xforms:trigger
ref="submitCredit">
<xforms:label>Issue
Credit</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="@makeCreURI"
value="concat('doBillingAction.php?
action=credit&acct=',../../acctId,'&amount=',../../totalDue)"/>
<xforms:load
ref="@makeCreURI"
show="new"/>
</xforms:action>
</xforms:trigger>
          </td>
          <td
align="center"
width="10%">
<xforms:textarea
ref="paymentAmt"/>
          </td>
          <td
align="center"
width="15%">
<xforms:trigger
ref="pay">
<xforms:label>Record</xforms:label>
```


© Copyright IBM Corporation 2007. All rights reserved.

```
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="@makeRctURI"
value="concat
('doBillingAction.php?
action=makepayment&acct=',../../acctId,'&amount=',../../paymen
tAmt,'&bal=',
../../totalDue -
../../paymentAmt)"/>
<xforms:insert
nodeset="../payment"
at="count
(../payment)"
position="after"/>
<xforms:setvalue
ref="../payment[count
(../payment)]/paymentDate"
value="substring(now(),1,10)"/>
<xforms:setvalue
ref="../payment[count
(../payment)]/amount"
value="../../paymentAmt"/>
<xforms:setvalue
ref="../payment[count
(../payment)]/acctId"
value="../../acctId"/>
<xforms:load
ref="@makeRctURI"
show="new"/>
<xforms:setvalue
ref="../paymentAmt"
value="0"/>
</xforms:action>
</xforms:trigger>
</td>
          <td
align="center"
width="10%">
<xforms:trigger
ref="bill">
<xforms:label>Bill</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="@makeBilURI"
value="concat('doBillingAction.php?
action=createbill&acct=',../../acctId,'&amount=',../../payment
Amt,'&bal=',../../totalDue)"/>
<xforms:load
ref="@makeBilURI"
show="new"/>
</xforms:action>
</xforms:trigger>
          </td>
        </tr>
      </table>
</xforms:repeat>

<xforms:bind
nodeset="instance('accts')//bill"
relevant="../accountState
!= 'Paid' and
../accountState
!= 'Current' and
../accountState
!= 'Credit'"/>
<xforms:bind
nodeset="instance('accts')//pay"
```


© Copyright IBM Corporation 2007. All rights reserved.

```
relevant="../accountState
!= 'Paid' and
../accountState
!=
'Credit'"/>
<xforms:bind
nodeset="instance('accts')//paymentAmt"
relevant="../accountState
!= 'Paid' and
../accountState
!=
'Credit'"/>
<xforms:bind
nodeset="instance('accts')//submitCollection"
relevant="../accountState
= 'Delinquent'"/>
<xforms:bind
nodeset="instance('accts')//submitCredit"
relevant="../accountState
= 'Credit'"/>
<xforms:bind
nodeset="instance('accts')//accountState"
calculate="
if(number(../totalDue)>0,
if(days-from-date(now())
-
if(../maxYearPaid='NaN',
days-from-date
(../creationDate),
days-from-date
(../lastPaymentDate))
> 30,
if(days-from-date(now())
-
if(../maxYearPaid='NaN',
days-from-date
(../creationDate),
days-from-date
(../lastPaymentDate))
> 90,
if(days-from-date(now())
-
if(../maxYearPaid='NaN',
days-from-date
(../creationDate),
days-from-date
(../lastPaymentDate))
> 180,
'Delinquent',
'Pastdue'),
'Due'),
'Current'),
if(number(../totalDue)
=
0,'Paid','Credit'))"/>
```

Next you'll learn about recording payments for accounts.

Recording payments

The recording process requires a receipt and the storage of the data. The billing tool handles this situation by opening a new window where the required data appears in a format suitable for printing and mailing. With equivalent ease, the customer could

© Copyright IBM Corporation 2007. All rights reserved.

receive an e-mail or any one of a number of procedural processes. The Record, Bill, and Collections buttons all behave similarly by manually formatting a GET submission activity and using that result as the argument for load with an attribute that appears first in this series at this point. The show=new attribute tells the browser to create a new viewing construct. Figure 8 shows the results of activating the COLLECT! button.

Figure 8. The collections display

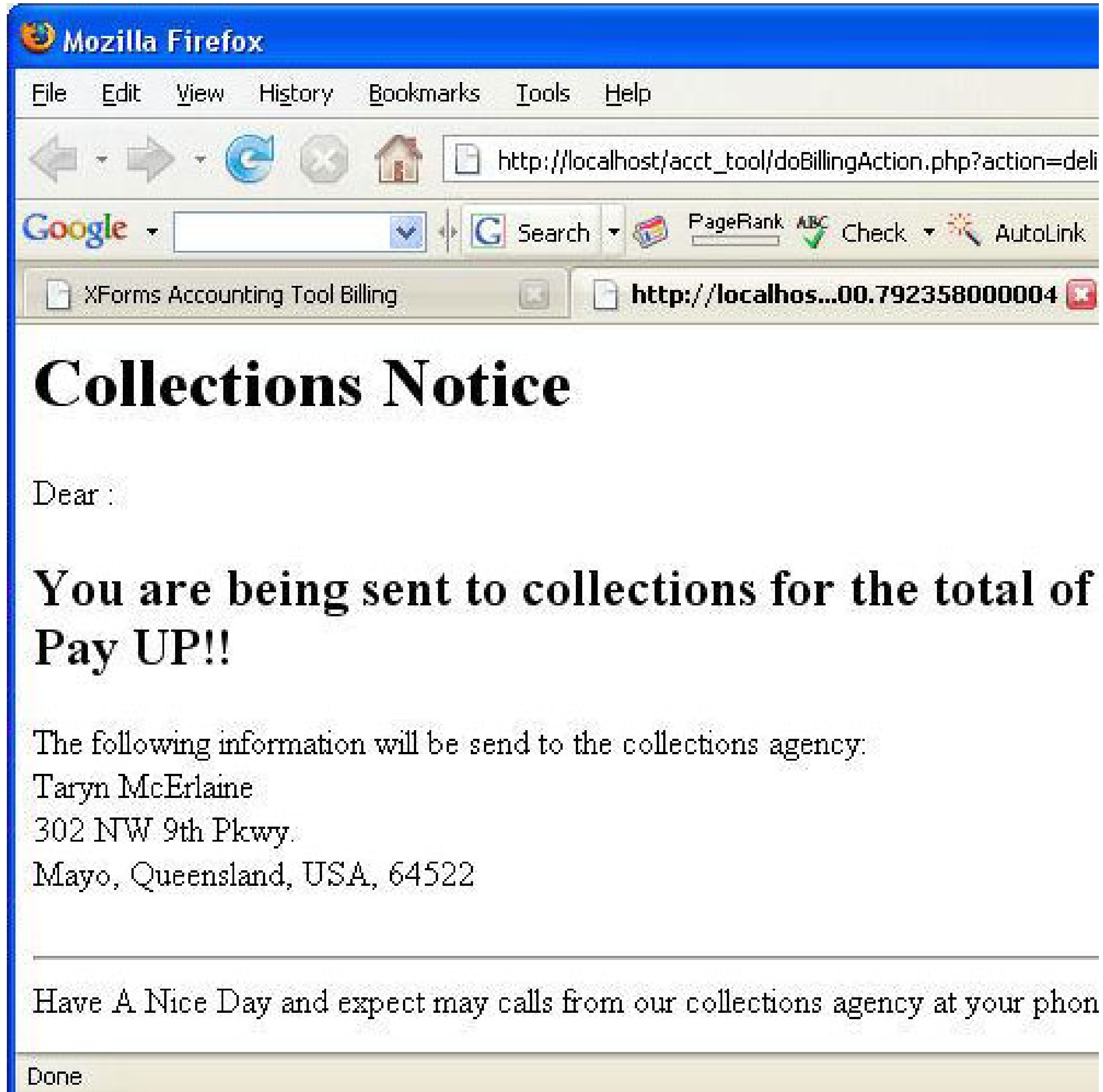
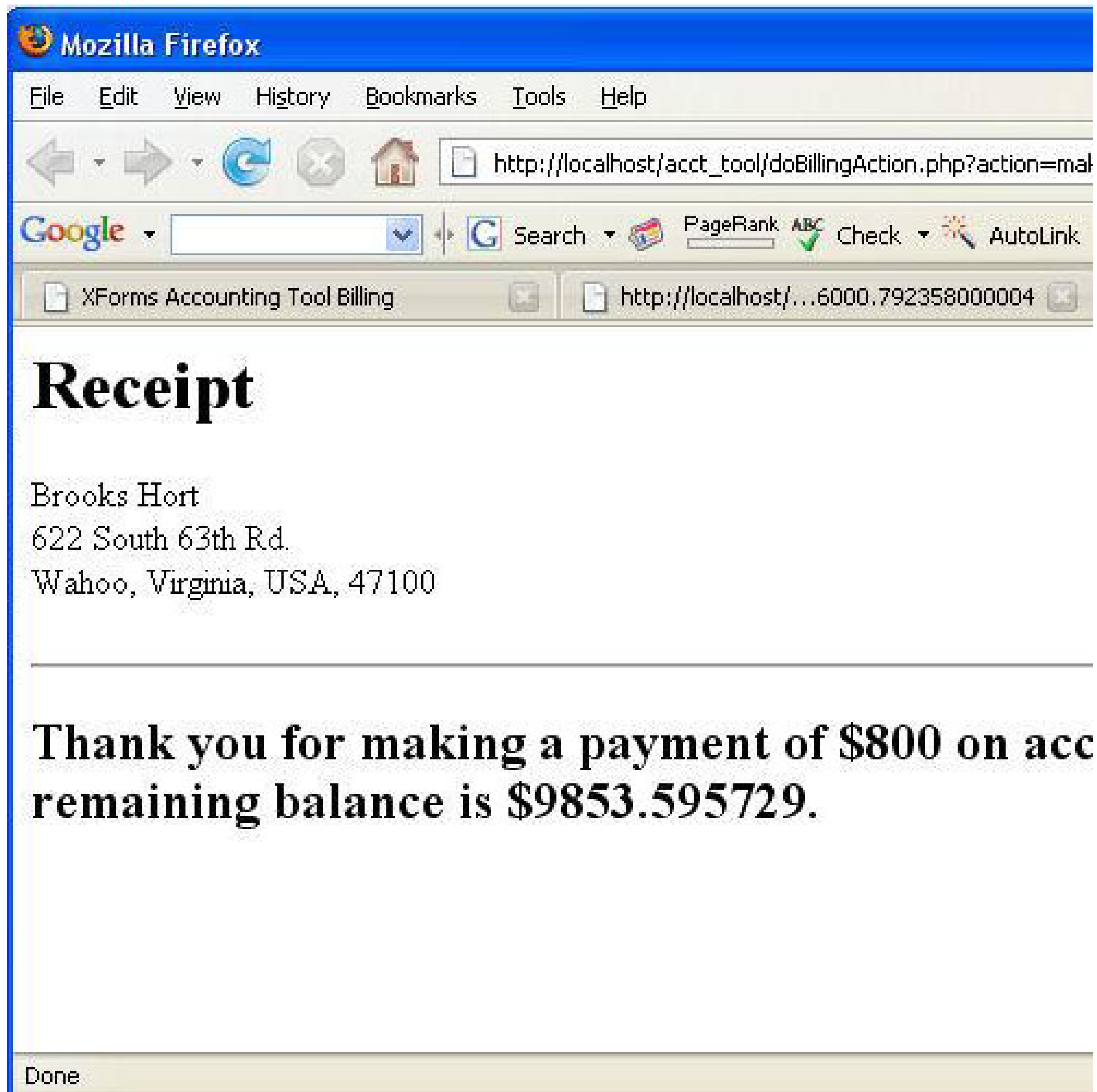


Figure 9 shows the result of activating the **Record** button.

Figure 9. The receipt display

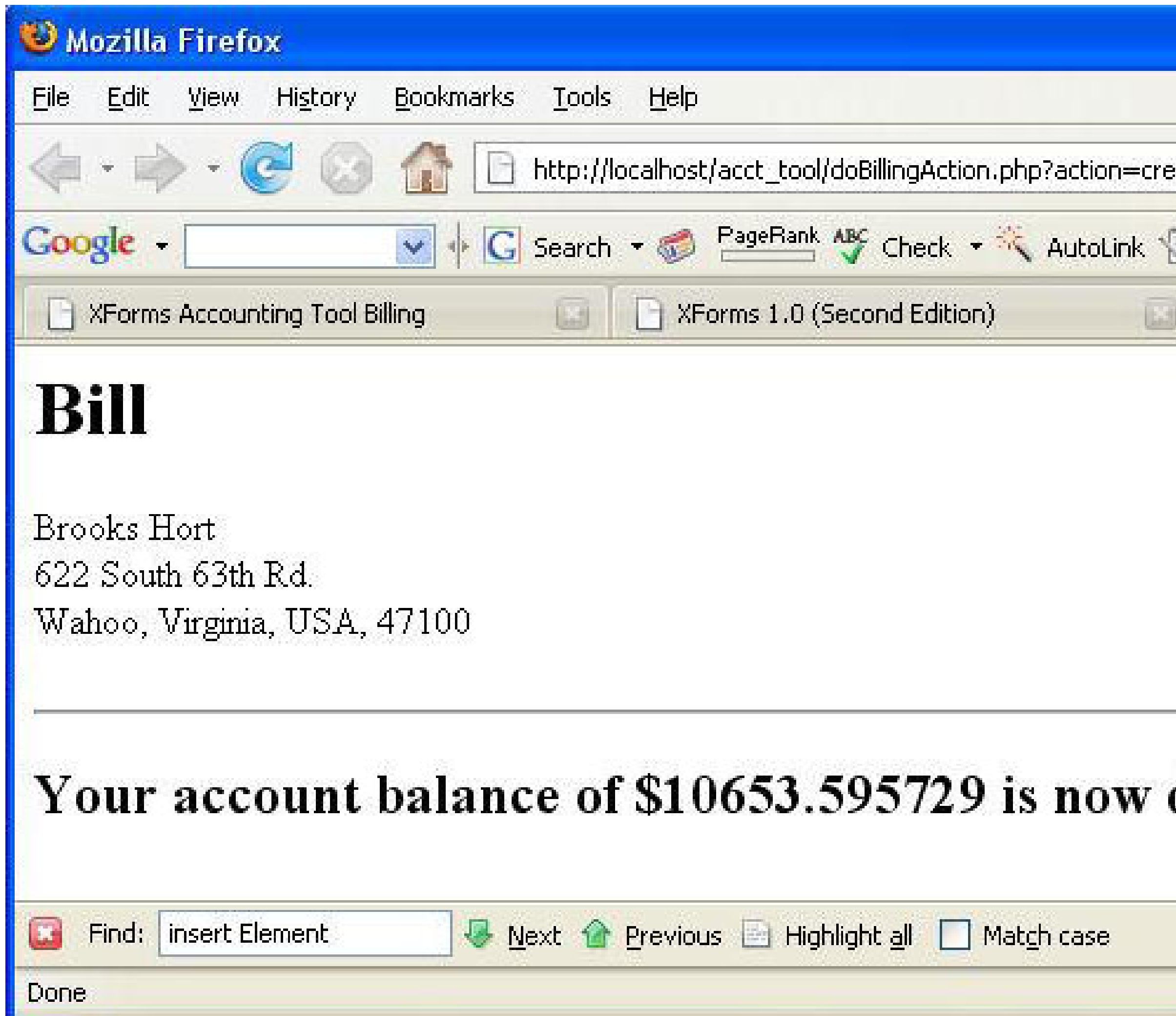
© Copyright IBM Corporation 2007. All rights reserved.



Activating the Bill button gives the results shown in Figure 10.

Figure 10. The bill display

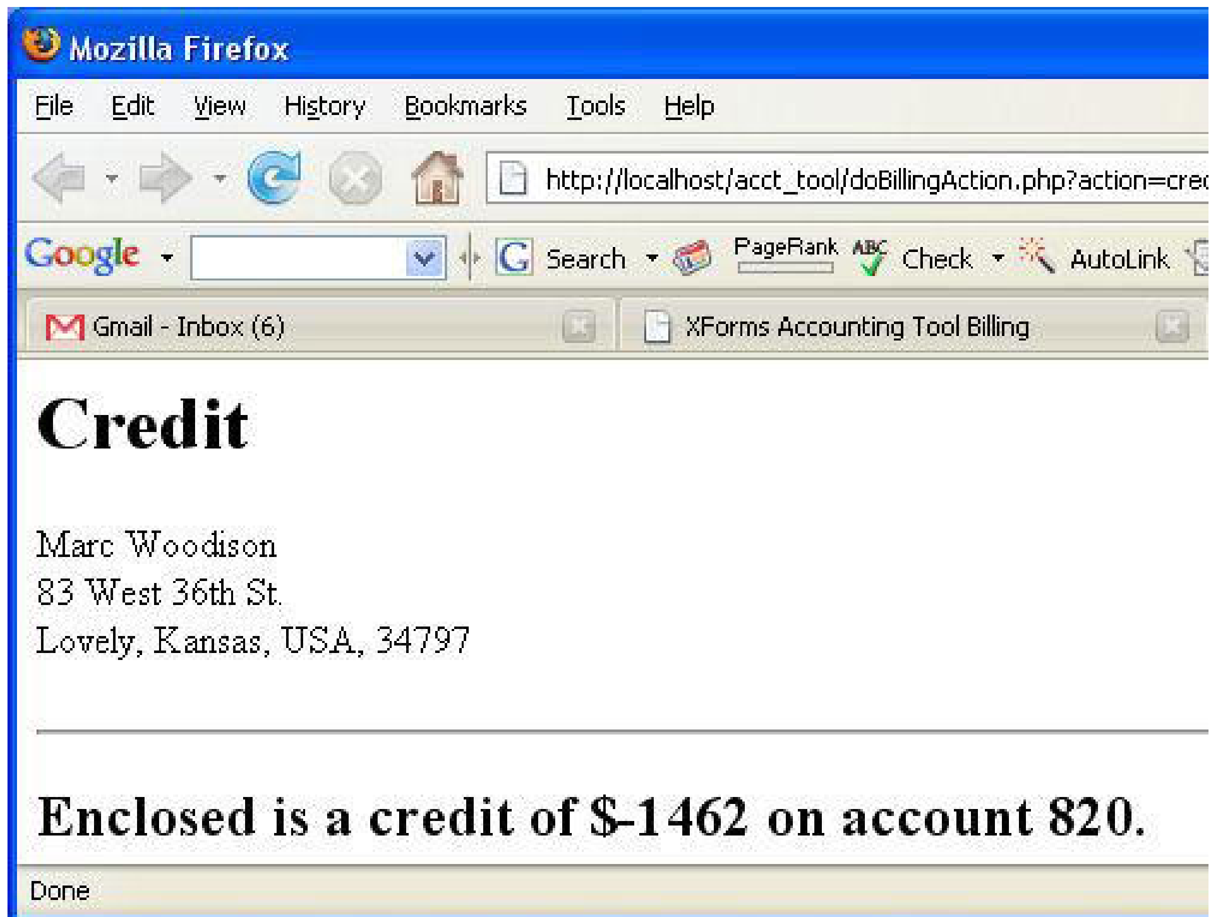
© Copyright IBM Corporation 2007. All rights reserved.



Selecting the **Issue Credit** button gives the results shown in Figure 11.

Figure 11. The credit display

© Copyright IBM Corporation 2007. All rights reserved.



Next, you'll see how to set the various states of an account based on the date of the last payment.

Calculating possible states for each account

Three different states appear in Figure 7. Each state creates different conditions for display. Accounts that have a balance greater than zero and have not been paid on for more than 180 days have the COLLECT! button displayed, while accounts that have zero or less as a balance have no buttons at all, though a negative balance would ultimately require a Credit button. Listing 19 contains the relevant statements and the calculate statement that create this effect.

Listing 19. Account state calculations

```
<xforms:bind
nodeset="instance('accts')//bill"
relevant="../accountState
!= 'Paid' and
../accountState
!= 'Current' and
```


© Copyright IBM Corporation 2007. All rights reserved.

```
../accountState
!= 'Credit'"/>
<xforms:bind
nodeset="instance('accts')//pay"
relevant="../accountState
!= 'Paid' and
../accountState
!= 'Credit'"/>
<xforms:bind
nodeset="instance('accts')//paymentAmt"
relevant="../accountState
!= 'Paid' and
../accountState
!= 'Credit'"/>
<xforms:bind
nodeset="instance('accts')//submitCollection"
relevant="../accountState
= 'Delinquent'"/>
<xforms:bind
nodeset="instance('accts')//submitCredit"
relevant="../accountState
= 'Credit'"/>
<xforms:bind
nodeset="instance('accts')//accountState"
calculate="
if(number(../totalDue)>0,
if(days-from-date(now())
-
if(../maxYearPaid='NaN',
days-from-date(../creationDate),
days-from-date(../lastPaymentDate))
> 30,
if(days-from-date(now())
-
if(../maxYearPaid='NaN',
days-from-date(../creationDate),
days-from-date(../lastPaymentDate))
> 90,
if(days-from-date(now())
-
if(../maxYearPaid='NaN',
days-from-date(../creationDate),
days-from-date(../lastPaymentDate))>180,
'Delinquent',
'Pastdue'),
'Due'),
'Current'),
if(number(../totalDue)
=
0,'Paid','Credit'))"/>
```

A new XPath function appears in this segment of code, the `days-from-date` function, which transforms a date value into an integral value referenced from the epoch. That function allows calculations like this one to occur. As Listing 19 shows, the escalation flows from a Paid account to a Delinquent account based upon a positive balance and passing 30, 90, and 180 days, and if no payments exist (meaning no payment dates exist), then the creation date is used for the comparisons. Another interesting feature of Listing 19 is the *and* operator. The Bill button applies to accounts that are due and greater. By performing two Boolean comparisons, that affect is achieved. The Boolean *and* operator is the textual *and*, not the & or && many programmers are familiar with.

© Copyright IBM Corporation 2007. All rights reserved.

How the numbers crunch

Behind the scenes, XForms performs a great amount of number crunching. The basic data structure consists of an account node with a set of nodes named *order* and a set of nodes named *payment*. By summing the nodeset order/amount, the total amount of money the customer has ever agreed to pay to the company is available. First, however, the XForms engine must calculate the amount of each order. Each order's total is calculated by the equation in Listing 20.

Listing 20. Order amount and total due

```
<xforms:bind
nodeset="instance('accts')//order/amount"
calculate="../quantity
* ../unitPrice *
(1-../discount)"/>
<xforms:bind
nodeset="instance('accts')//totalDue"
calculate="sum(..//order/amount)
-
sum(..//payment/amount)"/>
```

Listing 20 demonstrates an internal capability of XForms processors that offers some relief to the programmer. The browser processing the XForms code resolves the dependencies in the appropriate order. In this case, the order/amount nodes needed to be resolved before the totalDue node. The relevant property in Listing 19 demonstrates a similar behavior with the dependency of the relevant on the calculate for the account state.

The account state calculation requires dates. A couple of options for providing that information exist. One option is to use the PHP to extract the dates required, and another option is to have the data sorted. XForms 1.0, unfortunately, does not provide a `min` function that correctly handles dates. The code in Listing 21, however, demonstrates a more intensive approach to finding the minimum date from a list. The process involves ripping each date field apart into the constituent parts to enable the comparison the re-assembling the result into a date.

Listing 21. Finding the minimum order date (creation date) and maximum payment date (last payment date)

```
<xforms:bind
nodeset="instance('accts')//order/yearCreate"
calculate="substring(../orderDate,
1, 4)"/>
<xforms:bind
nodeset="instance('accts')//minYearCreate"
calculate="min(..//yearCreate)"/>
<xforms:bind
nodeset="instance('accts')//order/monthCreate"
calculate="if(../yearCreate
```


© Copyright IBM Corporation 2007. All rights reserved.

```
=
../../minYearCreate,
number(substring(../orderDate,
6, 2)),
13)"/>
<xforms:bind
nodeset="instance('accts')//minMonthCreate"
calculate="min(..//order/monthCreate)"/>
<xforms:bind
nodeset="instance('accts')//order/dayCreate"
calculate="if(../monthCreate
=
../../minMonthCreate,
number(substring(../orderDate,
9, 2)),
32)"/>
<xforms:bind
nodeset="instance('accts')//minDayCreate"
calculate="min(..//order/dayCreate)"/>
<xforms:bind
nodeset="instance('accts')//creationDate"
calculate="concat(../minYearCreate,
'-',
if(../minMonthCreate
< 10,
concat('0',../minMonthCreate),
../minMonthCreate),
'-',
if(../minDayCreate
< 10,
concat('0',../minDayCreate),
../minDayCreate))"/>

<xforms:bind
nodeset="instance('accts')//payment/yearPaid"
calculate="substring(../paymentDate,
1, 4)"/>
<xforms:bind
nodeset="instance('accts')//maxYearPaid"
calculate="max(..//yearPaid)"/>
<xforms:bind
nodeset="instance('accts')//payment/monthPaid"
calculate="if(../yearPaid
=
../../maxYearPaid,
number(substring(../paymentDate,
6, 2)),
0)"/>
<xforms:bind
nodeset="instance('accts')//maxMonthPaid"
calculate="max(..//payment/monthPaid)"/>
<xforms:bind
nodeset="instance('accts')//payment/dayPaid"
calculate="if(../monthPaid
=
../../maxMonthPaid,
number(substring(../paymentDate,
9, 2)),
0)"/>
<xforms:bind
nodeset="instance('accts')//maxDayPaid"
calculate="max(..//payment/dayPaid)"/>
<xforms:bind
nodeset="instance('accts')//lastPaymentDate"
calculate="concat(../maxYearPaid,
'-',
if(../maxMonthPaid
< 10,
concat('0',../maxMonthPaid),
```


© Copyright IBM Corporation 2007. All rights reserved.

```
../maxMonthPaid),
'-',
if(../maxDayPaid
< 10,
concat('0',../maxDayPaid),
../maxDayPaid))"/>
```

Next, you'll gain some insight on how to improve the load time of your XForm for large numbers of instance data.

Updating the numbers without the wait

An obvious improvement to the current billing XForms form is the addition of a limit on the number of accounts displayed at one time, so the browser does not have to maintain a data structure of the entire database. Having the entire database in the data structure emphasizes one important fact, though. It takes a significant amount of time to query, build, and load the data structure and that activity is costly to user patience. This form implements a shortcut to increase the responsiveness of the form. The record action contains an insert block similar to the insert described in the budgeting form. The code in question appears in Listing 18.

The insert code creates a new payment node and stores the pertinent data in that node. Altering the internal data structure does not alter the database; however, the PHP that generates the receipt can easily access the database that yields a much faster response than submitting a storage of the entire nodeset of accounts, or even just a single node, which would then require a reload of the entire data structure. Due to the calculation equations in Listing 20, the balance for that account immediately reflects the new information.

You're almost there! In the last portion of this section you'll learn how PHP helps the billing form

What the PHP does for the billing XForms form

PHP serves as an interface the SQL database throughout this series with most of the work performed by the XForms processor. One limitation of XForms, however, is the requirement that a node be present. XForms does not create new nodes with ease and it requires a template when a new node is created. Most of the data behind the billing XForm is calculated data and storing calculated data consumes valuable disk space. The PHP creates the account nodeset and nests the order and payment nodes below it, adding nodes to the order nodes for amount and to account for calculations as well. The code for performing this function is very similar to the code shown for calculating the budget nodes except the resultant data structure has less hierarchy.

© Copyright IBM Corporation 2007. All rights reserved.

Way to go! You've completed the tutorial. Now check out the summary to learn more about what's coming next, in Part 4.

Section 4. Summary

This tutorial introduced several concepts with XForms. The tutorial covered opening new browser windows and nested repeats, as well as addresses the mechanics of insert. The budgeting form introduced the process for removing nodes and the internal requirements of the data structures to enable displays. Indepth calculations, dependency chains, and several new XPath functions appeared in both the billing and budgeting forms.

Future installments of this series refine both the XForms implementations as necessary and the skills involved in XForms implementation. Next, Part 4 implements asset management and some more executive tools.

© Copyright IBM Corporation 2007. All rights reserved.

Downloads

Description	Name	Size	Download method
Sample code for this tutorial	accttool_part3_code.zip	100KB	HTTP

Information about download methods

© Copyright IBM Corporation 2007. All rights reserved.

Resources

Learn

- Learn the essentials for creating the next generation of forms in the following article: XForms basics (developerWorks, September 2006).
- Get some breadth to your XForms knowledge by viewing the following XForms series: Introduction to XForms, Part 1: The new Web standard for forms , Introduction to XForms, Part 2: Forms, models, controls, and submission actions, and Introduction to XForms, Part 3: Using actions and events (developerWorks, September 2006).
- Get the The PHP Manual.
- Learn more about XForms submission events in XForms tip: Using form submission events (developerWorks, November 2006).
- Find out how to accept XForms data in Java (developerWorks, October 2006), Perl (developerWorks, October 2006), and PHP (developerWorks, October 2006).
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- Learn all about XML and more about XForms in the IBM developerWorks XML zone.

Get products and technologies

- The XForms Recommendation is maintained by the W3C.
- Get MozzIE, an open-source control that allows you to render XForms in Internet Explorer.

Discuss

- Participate in the discussion forum for this content.
- developerWorks blogs: Get involved in the developerWorks community.

About the author

© Copyright IBM Corporation 2007. All rights reserved.

Stony Yakovac

Stony Yakovac is an engineer and freelance author living in Lava Hot Springs, Idaho. He works on a wide variety of projects, including software and digital hardware designs.

© Copyright IBM Corporation 2007. All rights reserved.