

# Use XForms to create an accounting tool, Part 4: More asset management and reporting

## Track user requests regarding assets in X-Trapolate

Skill Level: Intermediate

[Nicholas Chase \(ibmquestions@nicholaschase.com\)](mailto:ibmquestions@nicholaschase.com)

Freelance writer  
Backstop Media

[Stony Yakovac \(syakovac@gmail.com\)](mailto:syakovac@gmail.com)

Software engineer  
Freelance

17 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 4 of this six-part series demonstrates how to pull together many of the techniques touched on in earlier installments using the example of the order review form and the asset management form, with special privileges for procurement users. It also introduces new techniques for handling real-world issues.

## Section 1. Before you start

This tutorial is for developers investigating the use of XForms in real-world situations, rather than "toy" applications. It describes the use of XForms in creating two different accounting forms as part of the X-Trapolate accounting application. This tutorial assumes that you're familiar with the basics of XForms. No accounting knowledge is required.

## About this tutorial

View the other articles and tutorials in this [series](#).

In [Part 3](#) of this series, you created a form to track the assets of your fictional company in terms of budgets, profit, and expenses. Here in Part 4, you are going to look at assets a little more literally.

Every business has assets, whether they are buildings, assembly-line robots, computers, or simple tools. Those assets must be purchased, maintained, and replaced at various intervals. In this tutorial, you will build a form to track user requests regarding assets. You will build one form for typical users, and another for the procurement users responsible for resolving such issues.

You will also build a form for reviewing orders. In both cases, you will see some of the kinds of issues that arise when you attempt to build an actual application. How do you combine information from separate database tables? What happens when the data is not structured to make your XForms functionality convenient? This tutorial explores those issues.

In the course of this tutorial, you will learn:

- How to create a master-detail form using XForms
- How to use nested switch/case statements to handle multiple situations simultaneously
- How to build an XML instance from multiple database tables
- How to automatically set "live" data such as the current date after a form has loaded
- How to build a single-value checkbox

## 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

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 on 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.

## System requirements

The following software is required to follow along with this tutorial:

- 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. Overview

Over the first three parts of this series, you have been introduced to various concepts and techniques necessary for building a "real-world" application using XForms. In this tutorial, you are going to see how things interconnect as you build two forms from scratch, as well as the PHP backend necessary to make them work.

## Where we're going

This tutorial chronicles the building of two separate forms. The first is an order review form, which enables the user to view information regarding invoices, order dates, account numbers, and the description in quantities of the items purchased from the fictional furniture company that has been the subject of this series, as you can see in Figure 1.

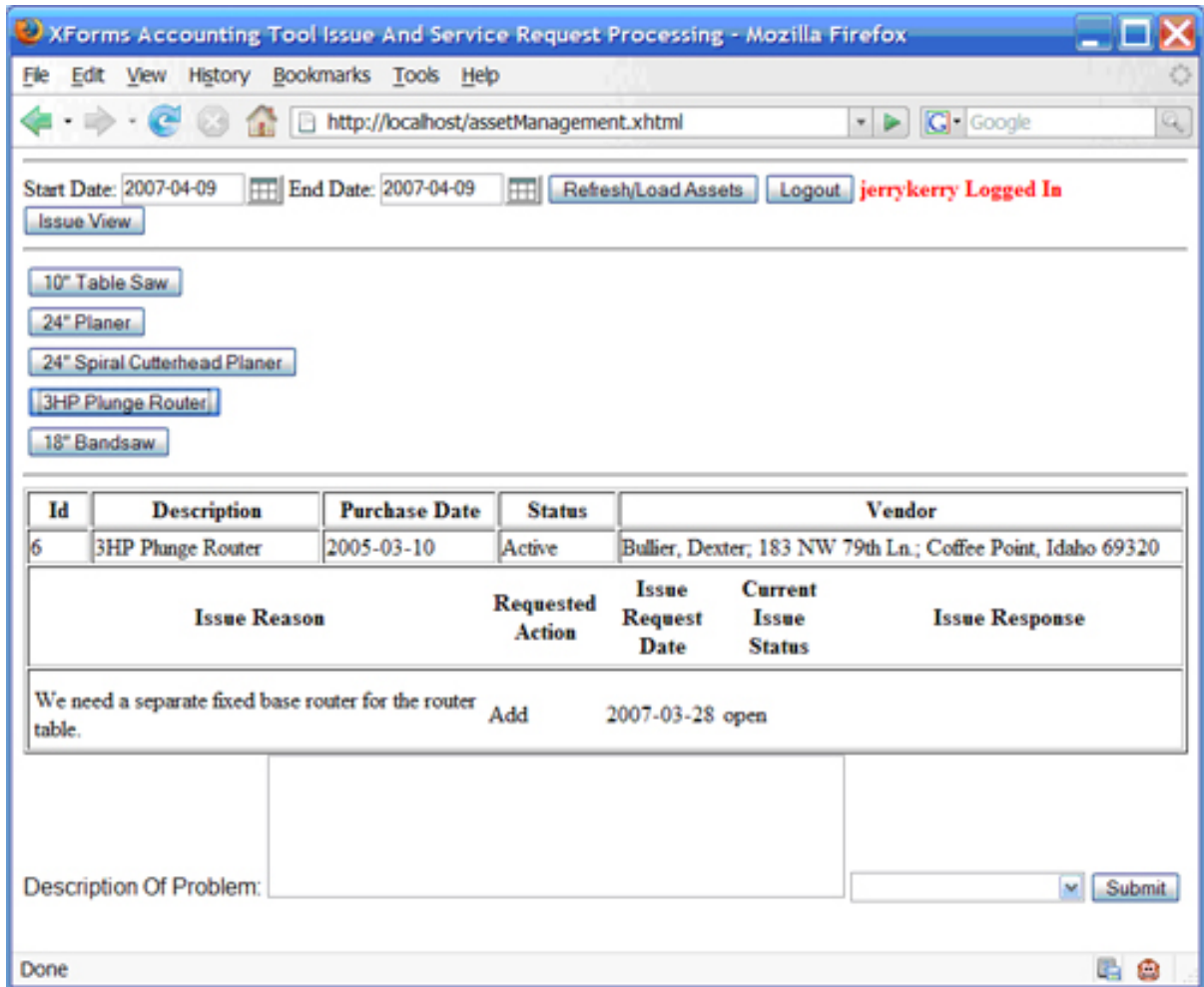
**Figure 1. The order review form**

Quantity Ordered	Description
929	Cocobolo 1in. Flatgrain 2ft. Square Cutting Board
988	Oriental Cocobolo 3ft. Round Coffee Table

This form acts as a "master detail" form. When the user chooses a new invoice, the information for that invoice automatically appears in the detail area below.

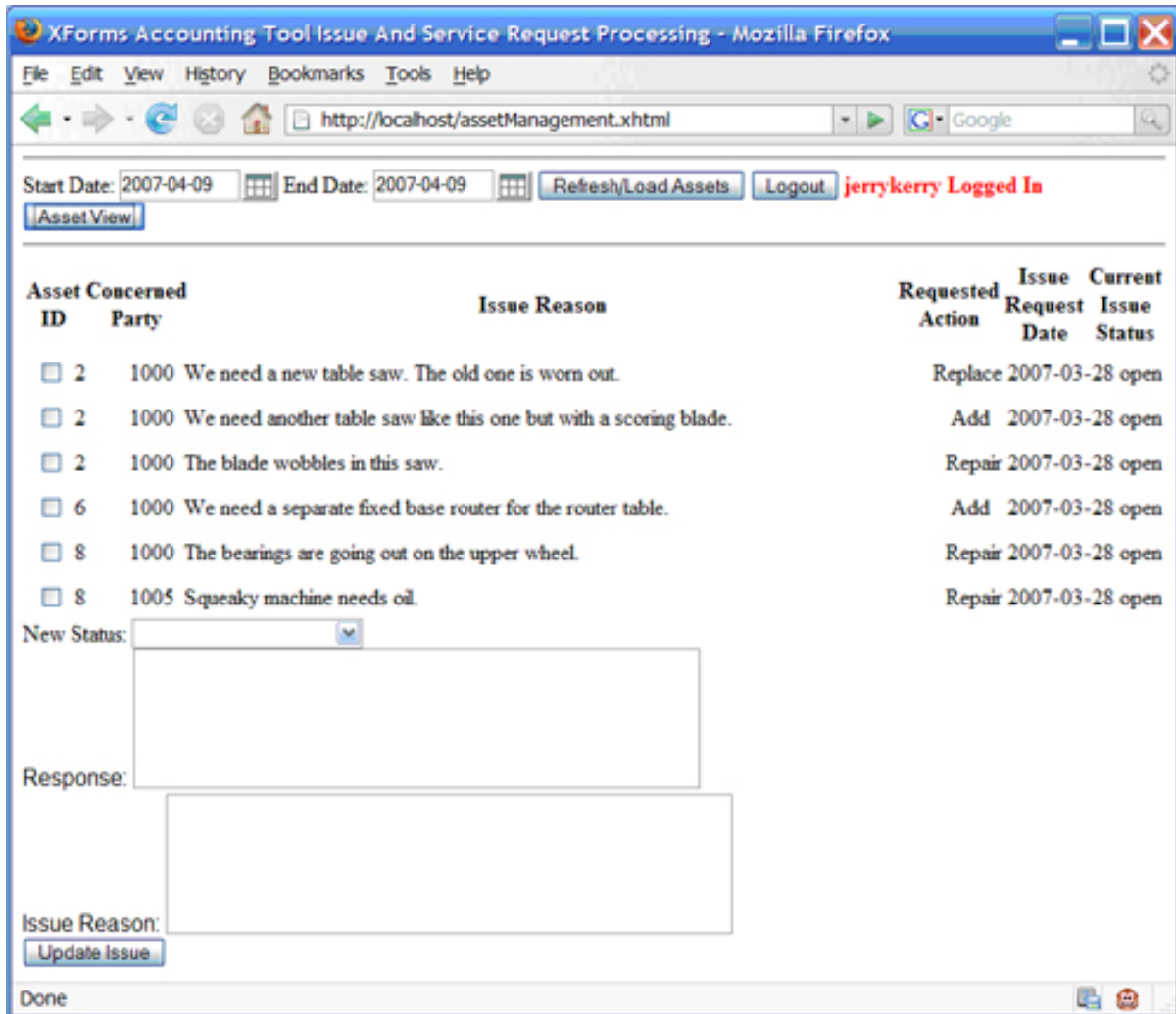
This tutorial also looks at building a more complex form, the asset management form, which tracks issues related to assets owned by the company, such as tools, as you can see in Figure 2.

**Figure 2. The asset management form**



This form also provides the second view for users who have procurement privileges, which enables them to manage issues directly, as you can see in Figure 3.

**Figure 3. The issue view**



## The database

To accomplish this task, you are going to need several new tables in your database. Table structures and sample data are included with the download file for this tutorial. If you're following along with the code, download the sample file, and run it against your database. You should, however, understand the way the tables you're going to see in this tutorial fit together. They are:

- invoices
- products
- departments
- assets
- assetdeptlinkage

- `assetissues`

The invoices table includes information on what was sold and how much of it was sold. It includes only the product identifier, or sku, so in order to display information detailing the actual products, you will need to extract it from the products table.

Previously, in [Part 2](#), all of the users were attached to a department. The `assetdeptlinkage` table associates departments with individual assets. Finally, the `assetissues` includes information on the actual issues and an asset identifier.

---

## Section 3. Invoices: The data

Let's start with the invoice form. The first step is to extract the data from the database using a PHP script and output it as XML to populate the instance within the form.

### The target structure

The instance form needs to include information on both the invoices and the items in them (see Listing 1).

#### Listing 1. The invoices instance document

```
<data
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <invoices>
    <invoice>
      <invoiceId/>
      <orderDate/>
      <acctId/>
      <order>
        <sku/>
        <quantity/>
        <unitPrice/>
        <description/>
      </order>
    </invoice>
    ...
  </invoices>
</data>
```

The structure identifies each individual item ordered on a single invoice as an `order`

element. The number of invoices included in this structure is determined by start and end date sent by the form.

## Process the input

The data comes in from the form as an XML instance with the structure seen in Listing 2.

### Listing 2. The argument instance

```
<formui xmlns="">
  <startDate/>
  <endDate/>
  <loadInvoices/>
  <errorMessage/>
</formui>
```

The PHP script loads this data as part of the POST request space (see Listing 3).

### Listing 3. Retrieving the parameters

```
<?php
  if
  (!isset($_HTTP_RAW_POST_DATA))
  $_HTTP_RAW_POST_DATA
  =
  file_get_contents("php://input");
  $xml =
  $_HTTP_RAW_POST_DATA;

  $args = new
  DomDocument('1.0');
  if ($xml)
  $args->loadXML($xml);

?>
```

Later, you will extract the individual parameters from the args document.

## Create the templates

The purpose of this script is to create an XML document you will return to the XForms form. In order to do that, you will first create the document to populate (see Listing 4).

### Listing 4. The document and templates

```
...
$args = new
```

```

DomDocument('1.0');
  if ($xml)
  $args->loadXML($xml);

  $doc = new
DomDocument('1.0');
$doc->loadXml('<?xml
version="1.0"
encoding="UTF-8"?>
<data
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <invoices/>
</data>');

  $iTplt =
$doc->createElement("invoice");
  $tmp =
$doc->createElement("invoiceId");
$iTplt->appendChild($tmp);
  $tmp =
$doc->createElement("orderDate");
$iTplt->appendChild($tmp);
  $tmp =
$doc->createElement("acctId");
$iTplt->appendChild($tmp);

  $oTplt =
$doc->createElement("order");
  $tmp =
$doc->createElement("sku");
$oTplt->appendChild($tmp);
  $tmp =
$doc->createElement("quantity");
$oTplt->appendChild($tmp);
  $tmp =
$doc->createElement("unitPrice");
$oTplt->appendChild($tmp);
  $tmp =
$doc->createElement("description");
$oTplt->appendChild($tmp);

$invoicesNode=$doc->GetElementsByTagName('invoices')->item(0);

echo
$doc->saveXML();
?>

```

The empty document is fairly straightforward. From there, use the document object to create the invoice and order elements. You will use these as templates to create populated elements in a moment. You will add these populated elements to the main `invoices` element, so retrieve a reference to that now.

Finally, you will output the document as the output of the script.

## Get the unique invoices

At this point, you have a choice. You're trying to populate the individual `invoice` elements, which potentially include multiple `order` elements. In a high-volume (or large data set) environment, you might choose to do this in one pass using a carefully crafted SQL statement. For this example, to keep the XForms generation

more straightforward, you will choose the other option, loading first the individual invoice information and then the orders.

Start by gathering the invoice information (see Listing 5).

### Listing 5. Gather the invoices

```

...
$invoicesNode=$doc->GetElementsByTagName('invoices')->item(0);

    $startDate=
    $args->GetElementsByTagName('startDate')
    ->item(0)->nodeValue;
    $endDate=
    $args->GetElementsByTagName('endDate')
    ->item(0)->nodeValue;

    $sqldb =
    mysql_connect('localhost',
    'root');
    if (!$sqldb)
        die('Could
    not connect to
    MySQL server at
    localhost
    because: ' .
    mysql_error());

    $sqlseldb =
    mysql_select_db('acct1',
    $sqldb);
    if
    (!$sqlseldb)
        die
    ('Can\'t select
    the acct database
    on MySQL server @
    ' .
    'localhost
    because: ' .
    mysql_error());

    $query =
    sprintf('SELECT
    DISTINCT
    invoiceId,
    orderDate, acctId
    ' .
    'FROM invoices ' .
    'WHERE orderDate
    >= \'%s\' ' .
    'AND orderDate <=
    \'%s\' ' .
    'ORDER BY
    invoices.orderDate
    DESC ;',
    $startDate,
    $endDate);

    $invoiceIdList
    =
    mysql_query($query,
    $sqldb);
    if
    (!$invoiceIdList)

```

```

        die
        ('Could not
        execute acct db
        query:
        <b>' . $query .
        '</b> because: '
        . mysql_error());

        while
        ($invoiceRow =
        mysql_fetch_assoc($invoiceIdList))
        {

            $newInvoice
            =
            $iTpl->cloneNode(TRUE);
            $invoicesNode->appendChild($newInvoice);
            $invoiceId
            =
            $invoiceRow['invoiceId'];
            for($temp =
            $newInvoice->firstChild;
            $temp
            != NULL;
            $temp =
            $temp->nextSibling)
            {

                if
                (array_key_exists($temp->nodeName,
                $invoiceRow)) {
                    $temp->nodeValue=$invoiceRow[$temp->nodeName];
                }
            }

        }

        echo
        $doc->saveXML();
        ?>

```

First, retrieve the start and end date information from the form, as represented in the args document. Next, connect to the server and select the appropriate database. From there, create the query, structuring it to retrieve the invoice-specific information for each relevant invoice, such as the `orderDate`.

### Avoid using user-submitted data directly

In a production application, you will want to avoid using user-submitted data directly, as shown here, to prevent SQL injection attacks. See [Resources](#) for more information on preventing this kind of security problem.

Once you have the list, loop through it. For each row, create a clone of the invoice template and appended to the invoices. Its child elements have been deliberately named the match those in the database, so you can loop through its child elements looking for appropriate values and populate them accordingly.

## Populate the orders



As you're processing each invoice, use the `invoiceId` to retrieve the individual orders. Clone the order template and populate it, appending it to the current invoice.

## Populate the product information

From a data standpoint, that's all you need, but in order to display the human-readable product description, you'll need to retrieve it from the database (see Listing 7).

### Listing 7. Retrieve the product information

```

...
        if
(array_key_exists($temp->nodeName,
$orderRow)) {
    $temp->nodeValue=$orderRow[$temp->nodeName];
    }

    }

    $query =
sprintf('SELECT *
FROM products
WHERE sku=%d ;',
$sku);

$productRows =
mysql_query($query,
$sqlldb);
    if
(! $productRows)
        die
('Could not
execute acct db
query:
<b>'. $query.
'</b> because: '
. mysql_error());
    while
($productRow =
mysql_fetch_assoc($productRows))
    {

for($temp =
$newOrder->firstChild;
$temp != NULL;
$temp =
$temp->nextSibling)
    {

        if
(array_key_exists($temp->nodeName,
$productRow)
&&
($temp->nodeValue
== NULL)) {
    $temp->nodeValue=$productRow[$temp->nodeName];
    }
    }
}

```

```
    }  
  }  
}  
  
echo  
$doc->saveXML();  
?>
```

At this point the document is complete, so let's look at the form.

---

## Section 4. Invoices: The master-detail form

The master-detail form, in which the user chooses from a list of identifiers and automatically gets the expanded information upon selection, is one of the most common forms in accounting, where individual items often need to be inspected. Let's look at one way to implement that in XForms.

Let's start by looking at the basic display of inventory information, and then move on to the master detail portion.

### The basic form

Start by creating the basic form, including the invoice information to be displayed (see Listing 8).

#### Listing 8. The basic form

```
<!DOCTYPE html  
PUBLIC  
"-//W3C//DTD  
XHTML 1.0  
Strict//EN"  
"http://www.w3.org/TR/xhtml1/D/tdxhtml1-strict.dtd">  
<html  
xmlns="http://www.w3.org/1999/xhtml"  
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"  
>  
  <head>  
  <title>XForms  
  Accounting Tool  
  Order  
  Viewer</title>  
  <xforms:model  
  id="atipm" >  
  
  <xforms:instance
```

```

id="atipi">
<formui xmlns="">
<startDate/>
<endDate/>
<loadInvoices/>
<errorMessage/>
</formui>
</xforms:instance>

<xforms:instance
id="work" >
<invoices
xmlns="">
<backorder/>
<invoice><invoiceId>2560555</invoiceId>
<orderDate>2007-04-06</orderDate><acctId>634</acctId>
<order><sku>113896</sku><quantity>875</quantity>
<unitPrice>561.00</unitPrice><description>Laser
Detailed
Jatoba 4ft.x5ft.
Legal
Bookcase</description></order><order>
<sku>115965</sku><quantity>42</quantity>
<unitPrice>750.00</unitPrice><description>Carved
Zebrawood
Nightstand with
Drawer and
Magazine
Rack</description></order><order><sku>124167
</sku><quantity>382</quantity><unitPrice>2772.00
</unitPrice><description>Customized
Teak 4ft.
Square Pedestal
Table</description></order></invoice>
<date/>
</invoices>
</xforms:instance>

<xforms:instance
id="data" >
<data
xmlns="" />
</xforms:instance>

<xforms:bind
nodeset="instance('atipi')/startDate"
type="xsd:date"/>
<xforms:bind
nodeset="instance('atipi')/endDate"
type="xsd:date"/>
</xforms:model>

<xforms:action
ev:event="xforms-ready">
<xforms:setvalue
ref="instance('atipi')/startDate"
value="'2006-07-01'"/>
<xforms:setvalue
ref="instance('atipi')/endDate"
value="substring(now(),1,10)"/>
</xforms:action>

</head>

<body>
<hr/>
<xforms:output
class="errorMessage"
ref="instance('acctToolBillInst')/errorMessage"/>

```

```

<xforms:input
ref="instance('atipi')/startDate">
<xforms:label>Start
Date</xforms:label>
</xforms:input>
<xforms:input
ref="instance('atipi')/endDate">
<xforms:label>End
Date</xforms:label>
</xforms:input>

    <hr />

</body>
</html>

```

Starting at the top, the `atipi` instance contains the parameters for the invoice loader form. The `work` instance contains the individual invoice to be displayed in the detail section. The `data` instance will ultimately contain all of the invoices returned by the PHP script.

The `bind` statements tell the form that the start date and end date are date values, which causes them to be displayed with the calendar widget, as you will see in [Figure 4](#). The value of these two fields is automatically set when the form has completely loaded, thanks to the action.

Finally, you display any error messages that have been returned (eventually -- nothing has actually been submitted yet!) and the start and end date.

## Displaying the invoices

Actually displaying the invoice information is a simple matter of using the output and repeat controls, as you have already seen earlier in the series (see [Listing 9](#)).

### Listing 9. Display the invoice

```

...
<xforms:input
ref="instance('atipi')/endDate">
<xforms:label>End
Date</xforms:label>
</xforms:input>

    <hr />

<table
width="100%">
<tr><th
align="left">Invoice
ID:
<xforms:output
value="instance('work')/invoice/invoiceId"/>
</th></tr>
<tr><th
align="left">Order

```

```

Date:
<xforms:output
value="instance('work')/invoice/orderDate"/>
</th></tr>
<tr><th
align="left">Account
ID:
<xforms:output
value="instance('work')/invoice/acctId"/>
</th></tr>
</table>
<hr/>
<table
width="100%">
<tr>
<th
width="50px">Quantity
Ordered</th>
<th width="100px"
align="left">Description</th>
</tr>
</table>

<xforms:repeat
id="orderLineRpt "
nodeset="instance('work')/invoice/order">

<table
width="100%">
<tr>
<td
align="center"
width="50px">
<xforms:output
value="quantity"/>
</td>
<td align="left"
width="100px">
<xforms:output
value="description"
/>
</td>
</tr>
</table>

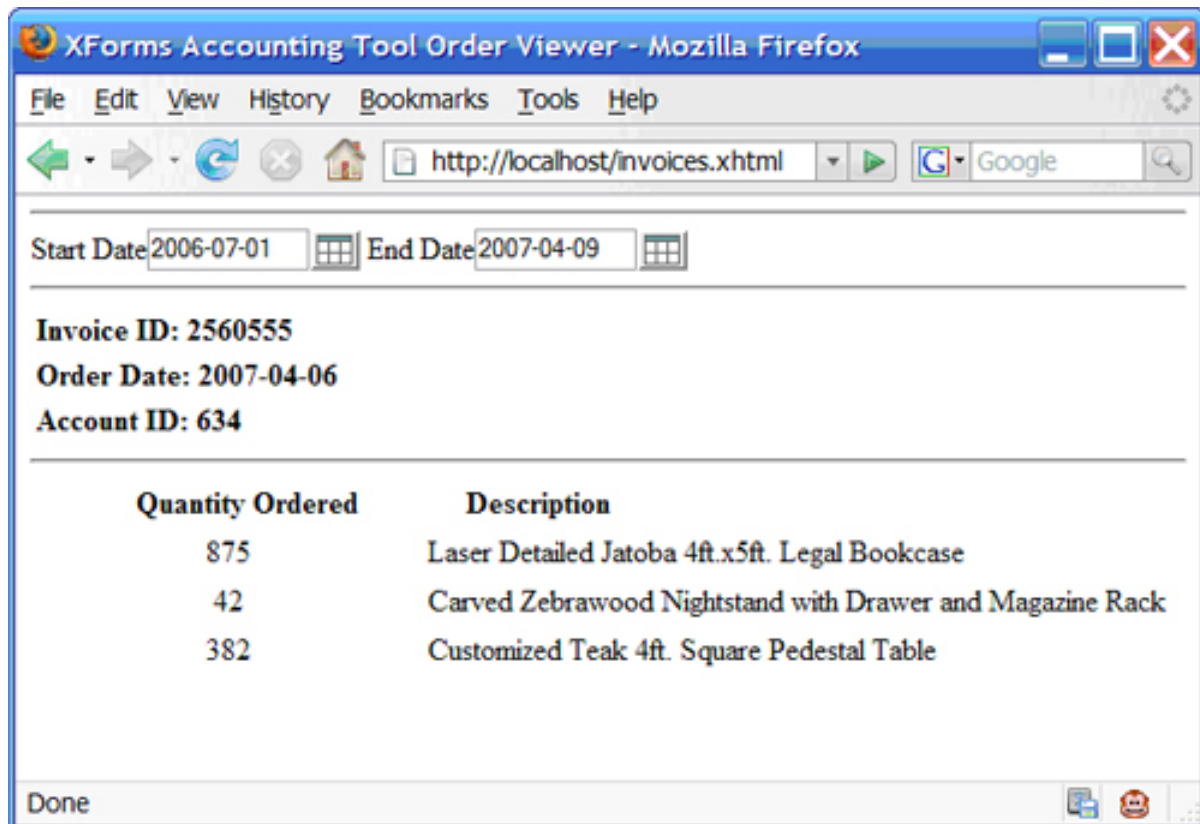
</xforms:repeat>

</body>
</html>

```

As you can see in Figure 4, everything is in place.

#### Figure 4. The invoice information



## Hiding the display

This form only appears on the day the invoice has been chosen, so let's use a switch/case construct to hide it until it's needed (see Listing 10).

### Listing 10. Hide the invoice structure

```

...
<xforms:input
ref="instance('atipi')/endDate">
<xforms:label>End
Date</xforms:label>
</xforms:input>

    <hr />

<xforms:switch>

<xforms:case
id="blank"
selected="true()">
</xforms:case>

<xforms:case
id="show_invoice_detail">
<table
width="100%">
<tr><th

```

```

align="left">Invoice
ID:
<xforms:output
value="instance('work')/invoice/invoiceId"/>
...
</tr>
</table>

</xforms:repeat>

</xforms:case>
</xforms:switch>

  </body>
</html>

```

When the form loads, the blank case is active, and the structure is hidden.

## Loading the invoices

With the display structure in place, it's time to load the live data (see Listing 11).

### Listing 11. Load the data

```

...
<xforms:instance
id="work" >
<invoices
xmlns="">
<backorder/>
<invoice/>
<date/>
</invoices>
</xforms:instance>

<xforms:instance
id="data" >
  <data
xmlns="" />
</xforms:instance>

<xforms:submission
id="load_invoices"
action="get_invoices.php"
method="post"
replace="instance"
instance="data"
ref="instance('atipi')"
/>

<xforms:bind
nodeset="instance('atipi')/startDate"
type="xsd:date"/>
...
<xforms:input
ref="instance('atipi')/endDate">
<xforms:label>End
Date</xforms:label>
</xforms:input>

<xforms:submit

```

```

submission="load_invoices"
ref="instance('atipi')/loadInvoices">
<xforms:label>Refresh/Load
Invoices</xforms:label>
<xforms:toggle
ev:event="DOMActivate"
case="show_invoice_detail"/>
</xforms:submit>

    <hr/>

<xforms:switch>
...

```

Here you are removing the sample data and adding a button to load the `data` instance (not the `work` instance) with the appropriate invoices as returned by the PHP script you wrote in the previous section. Also, when the user clicks the button, it changes the case to display the form you've just hidden.

At the moment, however, the form has no data.

## Choosing an invoice

Finally, you've arrived at the moment of truth, so to speak, and you need a way to select an individual invoice (see Listing 12).

### Listing 12. Select an invoice

```

...
<xforms:toggle
ev:event="DOMActivate"
case="show_invoice_detail"/>
</xforms:submit>

<xforms:select1
ref="instance('work')"
id="selinv"
incremental="true">
<xforms:label>Select
An Invoice To
View:
</xforms:label>
<xforms:itemset
nodeset="instance('data')/invoices/invoice">
<xforms:label
ref="invoiceId"/>
<xforms:copy
ref="."/>
</xforms:itemset>
</xforms:select1>

    <hr/>

<xforms:switch>
...

```

This is where the magic happens. The `select1` control contains an `itemset`,

which includes all of the `invoice` elements as its content. The actual `select1` element, however, is the `work` instance. It also carries the `incremental = "true"` attribute, which means that when the user selects a new value, the form reevaluates.

When the form reevaluates, the `select1` element takes the value of its contents, which in this case is the complete chosen invoice element. In this way, you're populating the `work` instance, which makes the data appear, as in Figure 5.

**Figure 5. The finished form**

XForms Accounting Tool Order Viewer - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/invoi

Start Date 2006-07-01 End Date 2007-04-09 Refresh/Load Invoices

Select An Invoice To View: 8605493

**Invoice ID: 8605493**  
**Order Date: 2007-04-04**  
**Account ID: 46**

Quantity Ordered	Description
929	Cocobolo 1in. Flatgrain 2ft. Square Cutting Board
988	Oriental Cocobolo 3ft. Round Coffee Table

Done

That completes the invoices form. Let's move on to the asset-management form.

---

## Section 5. Assets: The data

The asset-management form involves data from several different tables. In this

section, you will put together the PHP script to pull them together into a single XML instance.

## The target structure

The XML structure expected by the asset-management form it in many ways much like a relational database unto itself (see Listing 13).

### Listing 13. The expected XML data

```
<data>
  <assets>
    <asset>
<assetId/>
<description/>
<purchaseDate/>
<vendorId/>
      <issue>
<issueId/>
<reason/>
<action/>
<date/>
<status/>
<acctId/>
<response/>
      </issue>
    </asset>
  </assets>
  <vendors>
    <vendor
vendorId=" " >
<brief/>
<acctId/>
<company/>
<firstName/>
<lastName/>
<company/>
<street/>
      <city/>
<state/>
      <zip/>
<country/>
    </vendor>
  </vendors>
</data>
```

For each asset, you're going to want to display the vendor information, but it doesn't make sense to include all of that information with every asset. Instead, the instance includes the vendors separately, and references them by `vendorId`.

## Get the department

In this case, the form should only display information on assets that are associated with the current user's department, so you will need to check to make sure the user is logged in before doing anything else (see Listing 14).

## Listing 14. Checking login status

```
<?php
session_start();
if
(!isset($_HTTP_RAW_POST_DATA))
$_HTTP_RAW_POST_DATA
=
file_get_contents("php://input");
$xml =
$_HTTP_RAW_POST_DATA;

$args = new
DomDocument('1.0');
if ($xml)
$args->loadXML($xml);

if
($_SESSION['loginToken']
!=
$args->getElementsByTagName('loginToken')->item(0)->nodeValue)
{
    $errorMessage
=
$doc->getElementsByTagName('errorMessage')->item(0);
$errorMessage->nodeValue="Not
logged in!";
    exit;
}
$deptId =
$_SESSION['at_deptId'];

?>
```

Back in [Part 2](#), you built the login system so that it includes a login token assigned to the session when the user logs in. In this case, the script initializes the session, then checks the existing `loginToken` against the one submitted with the form. If it doesn't match, the script exits. If it does, the script retrieves the `departmentId` from the session.

## Create basic document and templates

The general process followed here is the same as the one used in the master detail form: Create an empty document and templates for the information being loaded (see [Listing 15](#)).

```
<?php
session_start();

$doc = new
DomDocument('1.0');
$doc->loadXml('<?xml
version="1.0"
encoding="UTF-8"?'>
```

```

<data
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">
<errorMessage/>
  <assets/>
  <vendors/>
</data>');

if
($_SESSION['loginToken']
!=
...
$deptId =
$_SESSION['at_deptId'];

$assetsNode =
$doc->getElementsByTagName('assets')->item(0);

$aTplt =
$doc->createElement("asset");
$assetsNode->appendChild($aTplt);
$temp =
$doc->createElement("assetId");
$aTplt->appendChild($temp);
$temp =
$doc->createElement("description");
$aTplt->appendChild($temp);
$temp =
$doc->createElement("purchaseDate");
$aTplt->appendChild($temp);
$temp =
$doc->createElement("vendorId");
$aTplt->appendChild($temp);

$iTplt =
$doc->createElement("issue");
$aTplt->appendChild($iTplt);
$iTplt->setAttribute("selected", '0');
$temp =
$doc->createElement("issueId");
$iTplt->appendChild($temp);
$temp =
$doc->createElement("reason");
$iTplt->appendChild($temp);
$temp =
$doc->createElement("action");
$iTplt->appendChild($temp);
$temp =
$doc->createElement("date");
$iTplt->appendChild($temp);
$temp =
$doc->createElement("status");
$iTplt->appendChild($temp);
$temp =
$doc->createElement("acctId");
$iTplt->appendChild($temp);
$temp =
$doc->createElement("response");
$iTplt->appendChild($temp);

$vendorsNode =
$doc->getElementsByTagName('vendors')->item(0);
$vTplt =
$doc->createElement("vendor");
$vendorsNode->appendChild($vTplt);
$temp =
$doc->createElement("brief");
$vTplt->appendChild($temp);

```

```

$temp =
$doc->createElement("acctId");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("company");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("firstName");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("lastName");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("company");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("street");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("city");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("state");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("zip");
$vTmpl->appendChild($temp);
$temp =
$doc->createElement("country");
$vTmpl->appendChild($temp);

echo
$doc->saveXML();

?>

```

In this case, however, you see this slight change. Whereas previously the templates were part of the document but not actually part of the structure -- sort of floating around in the ether -- in this case they are being appended to the actual assets node and vendors note.

## Get assets

The actual process of retrieving the asset information is also virtually identical to the one used for retrieving invoices (see Listing 16).

### Listing 16. Get the assets

```

...
$temp =
$doc->createElement("country");
$vTmpl->appendChild($temp);

$sqlldb =
mysql_connect('localhost',
'root');
if (!$sqlldb)
    die('Could not

```

```

connect to MySQL
server at
localhost
because: ' ' .
mysql_error());

$sqlselldb =
mysql_select_db('acct1',
$sqlldb);
if (!$sqlselldb)
    die ('Can\'t
select the acct
database on MySQL
server @ ' .
'localhost
because: ' ' .
mysql_error());

$queryEndDate=
$args->GetElementsByTagName('queryEndDate')->item(0)->nodeValue;
$queryStartDate=
$args->GetElementsByTagName('queryStartDate')->item(0)->nodeValue;

$query =
sprintf('SELECT
DISTINCT assetId
FROM
assetdeptlinkage
' .
'WHERE
deptId=%d', $deptId);

$assetIdRows =
mysql_query($query,
$sqlldb);
if
(!$assetIdRows)
    die ('Could
not execute acct
db query:
<b>' . $query .
'</b> because: ' '
. mysql_error());

while
($assetIdRow =
mysql_fetch_assoc($assetIdRows))
{
$assetId=$assetIdRow['assetId'];
    if
($queryEndDate !=
'') {
        $query =
sprintf(
'SELECT * FROM
assets WHERE ' .
'(assetId=%d AND
purchaseDate<=\'%s\'
AND ' .
'status=\'active\')',
$assetId,
$queryStartDate).
sprintf(' OR
(assetId= %d AND
purchaseDate<=\'%s\'
' .
'AND
retireDate>=\'%s\')',
$assetId,

```

```

$queryStartDate,
$queryEndDate);
    }
    else
    {
        $query =
sprintf('SELECT *
FROM assets WHERE
').
sprintf('status=\'active\'
AND assetId=%d',
$assetId);
    }

$assetQueryData =
mysql_query($query,
$sqlldb);
    if
(! $assetQueryData)
        die
('Could not
execute acct db
query:
<b>'. $query.
'</b> because: '
. mysql_error());
    while
($assetRow =
mysql_fetch_assoc($assetQueryData))
    {
        $newAsset=$aTmplt->cloneNode(TRUE);
        $assetsNode->appendChild($newAsset);
        $assetId =
        $assetRow['assetId'];
        for($temp=$newAsset->firstChild;
        $temp!=NULL;
        $temp=$temp->nextSibling)
        {

            foreach($assetRow
            as $field =>
            $value) {
                if($temp->hasAttribute($field))
                {
                    $temp->setAttribute($field,
                    $assetRow[$field]);
                }
            }
            if
            ($field ==
            'Vendor_acctId')
            $vendorAcctId=$value;
        }
        if
        ($temp->nodeName
        == 'vendorId')
        $temp->nodeValue=$vendorAcctId;

        if
        (array_key_exists($temp->nodeName,
        $assetRow)) {
            $temp->nodeValue=$assetRow[$temp->nodeName];
        }
    }
}

}

}

echo

```

```
$doc->saveXML();
?>
```

In this case, rather than using a fixed query, you need to adapt for the situation in which the user simply wants all active assets, versus those that had been purchased, but had not been retired between the start date and end date.

## Add vendor information

Once you have the asset, you need to add the vendor information, which is stored in the accounts table (see Listing 17).

### Listing 17. Adding the vendor information

```
...
        if
        (array_key_exists($temp->nodeName,
$assetRow)) {
            $temp->nodeValue=$assetRow[$temp->nodeName];
        }

        $query =
sprintf("SELECT *
FROM accounts
WHERE
acctId=%d;",
$vendorAcctId);
        $vendorRows
=
mysql_query($query,
$sqlldb);
        if
        (! $vendorRows)
            die
            ('Could not
execute acct db
query:
<b>'. $query.
'</b> because: '
. mysql_error());
        while
        ($vendorRow =
mysql_fetch_assoc($vendorRows))
        {
            $newVendor=$vTmpl->cloneNode(TRUE);
            $vendorsNode->appendChild($newVendor);
            for($temp =
            $newVendor->firstChild;
            $temp != NULL;
            $temp =
            $temp->nextSibling)
            {

                foreach($vendorRow
                as $field =>
                $value) {
                    if($temp->hasAttribute($field))
                    {
```

```

$temp->setAttribute($field,
$vendorRow[$field]);
    }
    }
    if
(array_key_exists($temp->nodeName,
$vendorRow)) {
$temp->nodeValue=$vendorRow[$temp->nodeName];
    }
    }
}
}

echo
$doc->saveXML();

?>

```

By now, the process should be familiar.

## Add issues

The same process applies for ending any issues associated with the asset (see Listing 18).

### Listing 18. Adding issues

```

...
    if
(array_key_exists($temp->nodeName,
$vendorRow)) {
$temp->nodeValue=$vendorRow[$temp->nodeName];
    }
    }
}

$query =
sprintf('SELECT *
FROM assetissues
WHERE
assetId=%d;',
$assetId);
$issueRows =
mysql_query($query,
$sqlldb);
if
(!$issueRows)
    die
('Could not
execute acct db
query:
<b>'. $query. '</b>'.
'because: ' .
mysql_error());
while
($issueRow =
mysql_fetch_assoc($issueRows))
{

```

```
        $newIssue =
        $iTpl->cloneNode(TRUE);
        $newIssue->setAttribute('real','1');
        for($temp =
        $newIssue->firstChild;
            $temp
            != NULL;
            $temp=$temp->nextSibling)
        {

        foreach($issueRow
        as $field =>
        $value) {
        if($temp->hasAttribute($field))
        {
        $temp->setAttribute($field,
        $issueRow[$field]);
        }
        }
        if
        (array_key_exists($temp->nodeName,
        $issueRow)) {
        $temp->nodeValue=$issueRow[$temp->nodeName];
        }
        }
        $newAsset->appendChild($newIssue);
    }
}

$assetsNode->removeChild($aTpl);
$vendorsNode->removeChild($vTpl);

echo
$doc->saveXML();

?>
```

Notice the very last step; the templates, if you recall, are added to the actual document. Because they are empty, you'll want to remove them before returning the document.

Now you're ready for the form.

---

## Section 6. The asset management form: Nested switch/cases

The asset management form is a fairly complex XForms form that enables the user to add information to the instance and to save it back to the database. It also makes use of nested switch/case statements to handle the problem of multiple conditions that are not necessarily related.

## Logging in (the XForm)

Start by creating a basic form, including the capability for the user to log in (see Listing 19).

### Listing 19. The login form

```
<!DOCTYPE html
PUBLIC
"-//W3C//DTD
XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/D/tdxhtml1-strict.dtd">
<html
xmlns="http://www.w3.org/1999/xhtml"
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"
>
  <head>
    <title>XForms
    Accounting Tool
    Issue And Service
    Request
    Processing</title>

    <link
    rel="stylesheet"
    href="style.css"
    type="text/css"/>
    <style
    type="text/css">
      textarea
      { font-family:
      sans-serif;
      height: 6em;
      width: 400px; }
    </style>

    <xforms:model
    id="atuim" >

    <xforms:instance
    id="isol">
      <isol
      xmlns="">
        <queryStartDate/>
        <queryEndDate/>
      </isol>
    </xforms:instance>

    <xforms:instance
    id="atuii">
      <formui xmlns="">
        <loginLogout>Login</loginLogout>
        <password/>
        <login>l</login>
        <loginToken/>
        <username/>
        <deptId/>
        <welcome/>

      <issueView/>
```

```

<errorMessage/>

<issueId/>
<statusOpts
txt="open"/>
<statusOpts
txt="closed"/>
<statusOpts
txt="pending"/>
<statusOpts
txt="confirmed"/>

<queryStartDate/>
<queryEndDate/>
<queryIssueStatus/>
<queryAcctId/>
<loadAssets/>

<numIssuesSelected/>
<someIssuesSelected/>
<issueMod/>
<issueReason/>
<issueAction/>
<issueStatus/>
<issueResponse/>
<issueSpecialRepairInstructions/>
<issueRepairInstructions/>
<issueDate/>
<issueAssetId/>
<updateIssue/>
<submitIssue/>
<issueActionOpts
txt="Repair"/>
<issueActionOpts
txt="Retire"/>
<issueActionOpts
txt="Replace"/>
<issueActionOpts
txt="Purchase"/>
<issueActionOpts
txt="Remove"/>
<issueActionOpts
txt="Add"/>
</formui>
</xforms:instance>

<xforms:instance
id="work" >
<workData
xmlns="" >
</workData>
</xforms:instance>

<xforms:instance
id="queryData" >
<data
xmlns="" >
<errorMessage/>
<assets/>
<vendors/>
</data>
</xforms:instance>

<xforms:submission
id="login_logout"
action="login_logout.php"
method="post"
replace="instance"

```

```

instance="atuii"
ref="instance('atuii')"
/>

<xforms:bind
nodeset="instance('atuii')/password"
relevant="number(instance('atuii')/login)=1"/>
<xforms:bind
nodeset="instance('atuii')/username"
relevant="number(instance('atuii')/login)=1"/>
<xforms:bind
nodeset="instance('atuii')/welcome"
relevant="instance('atuii')/loginLogout
= 'Logout'"
calculate="concat('Welcome
',
instance('atuii')/username)"/>
</xforms:model>

</head>

<body>
<hr/>

<xforms:input
ref="instance('atuii')/username">
<xforms:label>Username:
</xforms:label>
</xforms:input>
<xforms:secret
ref="instance('atuii')/password">
<xforms:label>Password:
</xforms:label>
</xforms:secret>
<xforms:submit
submission="login_logout"
ref="instance('atuii')/loginLogout">
<xforms:label
ref="instance('atuii')/loginLogout"/>
</xforms:submit>

<xforms:output
class="errorMessage"
value="concat(instance('atuii')/errorMessage,
instance('queryData')/errorMessage)"/>
<xforms:output
ref="instance('atuii')/welcome"/>

</body>
</html>

```

The `atuii` instance is long, but is a placeholder for all of the information needed by the actual user interface, such as the current department ID and in the information added to an asset for submission. The `work` instance holds the current asset as before, and the `queryData` instance holds the asset data returned from the PHP script.

The `bind` statements determine whether the form should display the username and password boxes or the welcome message and the logout button. When the user logs in, the form calls the `login_logout` submission, which calls the `login_logout.php` script.

## Logging in: The PHP

The login\_logout.php script does exactly what it sounds like: If users are not logged in, it logs them in. If they are, it logs them out (see Listing 20).

### Listing 20. The login\_logout.php script

```
<?php
session_start();
if
(!isset($_HTTP_RAW_POST_DATA))
$_HTTP_RAW_POST_DATA
=
file_get_contents("php://input");
$xml =
$_HTTP_RAW_POST_DATA;
$doc = new
DomDocument('1.0');
$doc->loadXML($xml);

if
($doc->GetElementsByTagName('loginLogout')->item(0)->nodeValue
== 'Logout'){
unset($_SESSION['loginToken']);
unset($_SESSION['username']);
unset($_SESSION['at_deptId']);
unset($_SESSION['acctId']);
$doc->GetElementsByTagName('deptId')->item(0)->nodeValue
= "";
$doc->GetElementsByTagName("errorMessage")->item(0)->nodeValue
=
sprintf('You have
successfully
logged out,
please reload');
$doc->GetElementsByTagName('loginLogout')->item(0)->nodeValue
=
"Login";
$doc->GetElementsByTagName("username")->item(0)->nodeValue
= "";
$doc->GetElementsByTagName("loginToken")->item(0)->nodeValue
= "";
$doc->GetElementsByTagName('login')->item(0)->nodeValue=1;
echo
$doc->saveXML();
exit;
}

if
(array_key_exists('loginToken',
$_SESSION)) {
$doc->GetElementsByTagName("username")->item(0)->nodeValue
=
$_SESSION['username'];
$doc->GetElementsByTagName("loginToken")->item(0)->nodeValue
=
$_SESSION['loginToken'];
$doc->GetElementsByTagName("errorMessage")->item(0)->nodeValue
=
sprintf('%s
Logged
In', $_SESSION['username']);
$doc->GetElementsByTagName('loginLogout')->item(0)->nodeValue
```

```

=
"Logout";
$doc->GetElementsByTagName('deptId')->item(0)->nodeValue
=
$_SESSION['at_deptId'];
$doc->GetElementsByTagName('login')->item(0)->nodeValue=0;
echo
$doc->saveXML();
exit;
}

$pass =
$doc->getElementsByTagName("password")->item(0)->nodeValue;
$username =
$doc->getElementsByTagName("username")->item(0)->nodeValue;

if ($username == ''
&& $pass == ''){
echo
$doc->saveXML();
exit;
}

$sqlldb =
mysql_connect('localhost',
'root');
if (!$sqlldb)
die('Could not
connect to MySQL
server at
localhost
because: ' .
mysql_error());

$sqlselldb =
mysql_select_db('acct1',
$sqlldb);
if (!$sqlselldb)
die('Can\'t
select the acct
database on MySQL
server @
localhost
because: ' .
mysql_error());

$sqlQuery =
sprintf("SELECT *
FROM accounts
WHERE
username='%s' ".
"and
password='%s'",
mysql_real_escape_string($username),
mysql_real_escape_string($pass));
$queryData =
mysql_query($sqlQuery,
$sqlldb);
if (!$queryData)
die('Could not
insert token into
tokentable on
MySQL server
because: '
. mysql_error());

$doc->getElementsByTagName("password")->item(0)->nodeValue
= '';

```

```

$numRows =
mysql_num_rows($queryData);
if($numRows <= 0)
{
mysql_close($sqldb);
$doc->getElementsByTagName("errorMessage")->item(0)->nodeValue
=
'Invalid login
credentials!';
echo
$doc->saveXML();
exit;
}
else
{
$row =
mysql_fetch_assoc($queryData);
$_SESSION['at_deptId']
= $row['deptId'];
$_SESSION['acctId']
= $row['acctId'];
$_SESSION['username']
= $uname;

$loginToken =
rand();
$_SESSION['loginToken']
= $loginToken;
$now =
localtime();
$datetime =
sprintf("%04d-%02d-%02d", $now[5]+1900, $now[4], $now[3]);
$sqlQuery =
sprintf("INSERT
INTO logintokens
".
"(logintoken,
username,
creation) ".
" VALUES ( '%d',
'%s', '%s');",
$_SESSION['loginToken'],
mysql_real_escape_string($username),
$datetime);
$queryData =
mysql_query($sqlQuery,
$sqldb);
if
(!$queryData) die
('Could not
insert token into
tokentable on '.
'MySQL server
because: ' .
mysql_error());
mysql_close($sqldb);

$doc->getElementsByTagName("password")->item(0)->nodeValue
= '';
$doc->GetElementsByTagName("errorMessage")->item(0)->nodeValue
=
sprintf('%s
Logged
In', $_SESSION['username']);
$doc->getElementsByTagName('loginLogout')->item(0)->nodeValue
=
"Logout";
$doc->GetElementsByTagName("loginToken")->item(0)->nodeValue
=

```

```

$_SESSION['loginToken'];
$doc->getElementsByTagName('deptId')->item(0)->nodeValue
=
$_SESSION['at_deptId'];
$doc->GetElementsByTagName('login')->item(0)->nodeValue=0;

    echo
$doc->saveXML();
    exit;
}
?>

```

This is actually fairly straightforward, taking things one step at a time. If the user is trying to log out, it clears the session and the instance, and sets it back to asking for login, rather than logout. (In an ideal world, the login form would then appear, but a bug in the current implementation prevents this, so here we ask the user to reload.)

Next, check to see if the user is already logged in (perhaps from interacting with another form) and if so, add the appropriate information to the instance and send it back to the form.

If the user is not already logged in, check for username and password -- return the data with no information if they are not present -- and then connect to the database and check their authentication. If their information is incorrect, it sends an error message back to the form. If it is correct, it extracts information such as the appropriate department ID and edit to the session. Also, it creates a new log inToken and edits to the session and the database. Finally, it populates the instance, setting it to ask for logout rather than login, and sends it back to the form.

## Load assets (the form)

Now that the user is logged in, it's time to load the assets (see Listing 21).

### Listing 21. Load the assets

```

...
replace="instance"
instance="atuii"
ref="instance('atuii')"
/>

<xforms:submission
id="load_assets"
action="get_assets.php"
method="post"
replace="instance"
instance="queryData"
ref="instance('atuii')"
/>

<xforms:bind
nodeset="instance('isol')/queryStartDate"
type="xsd:date"/>
<xforms:bind

```

```

nodeset="instance('isol')/queryEndDate"
type="xsd:date"/>

<xforms:bind
nodeset="instance('atuii')/queryStartDate"
calculate="instance('isol')/queryStartDate"/>
<xforms:bind
nodeset="instance('atuii')/queryEndDate"
calculate="instance('isol')/queryEndDate"/>

<xforms:bind
nodeset="instance('atuii')/password"
...
</xforms:model>

<xforms:action
ev:event="xforms-ready">
<xforms:setvalue
ref="instance('isol')/queryStartDate"
value="substring(now(),1,10)"/>
<xforms:setvalue
ref="instance('isol')/queryEndDate"
value="substring(now(),1,10)"/>
<xforms:dispatch
name="xforms-submit"
target="login_logout"/>
</xforms:action>
</head>

<body>
<hr/>

<xforms:input
ref="instance('isol')/queryStartDate">
<xforms:label>Start
Date:
</xforms:label>
</xforms:input>
<xforms:input
ref="instance('isol')/queryEndDate">
<xforms:label>End
Date:
</xforms:label>
</xforms:input>
<xforms:submit
submission="load_assets"
ref="instance('atuii')/loadAssets">
<xforms:label>Refresh/Load
Assets</xforms:label>
</xforms:submit>

<xforms:input
ref="instance('atuii')/username">
...

```

As before, you are using values that have been down to the date type, so a calendar widget appears. In this case, you're using an isolated instance and setting its values when the form loads. You can then drop those values into the "live" nodes using the `calculate` attribute.

Finally, add the Refresh/Load Assets button, which calls the `load_assets` submission. The submission refers to the `get_assets.php` script, which you built in the previous section.

## Display assets

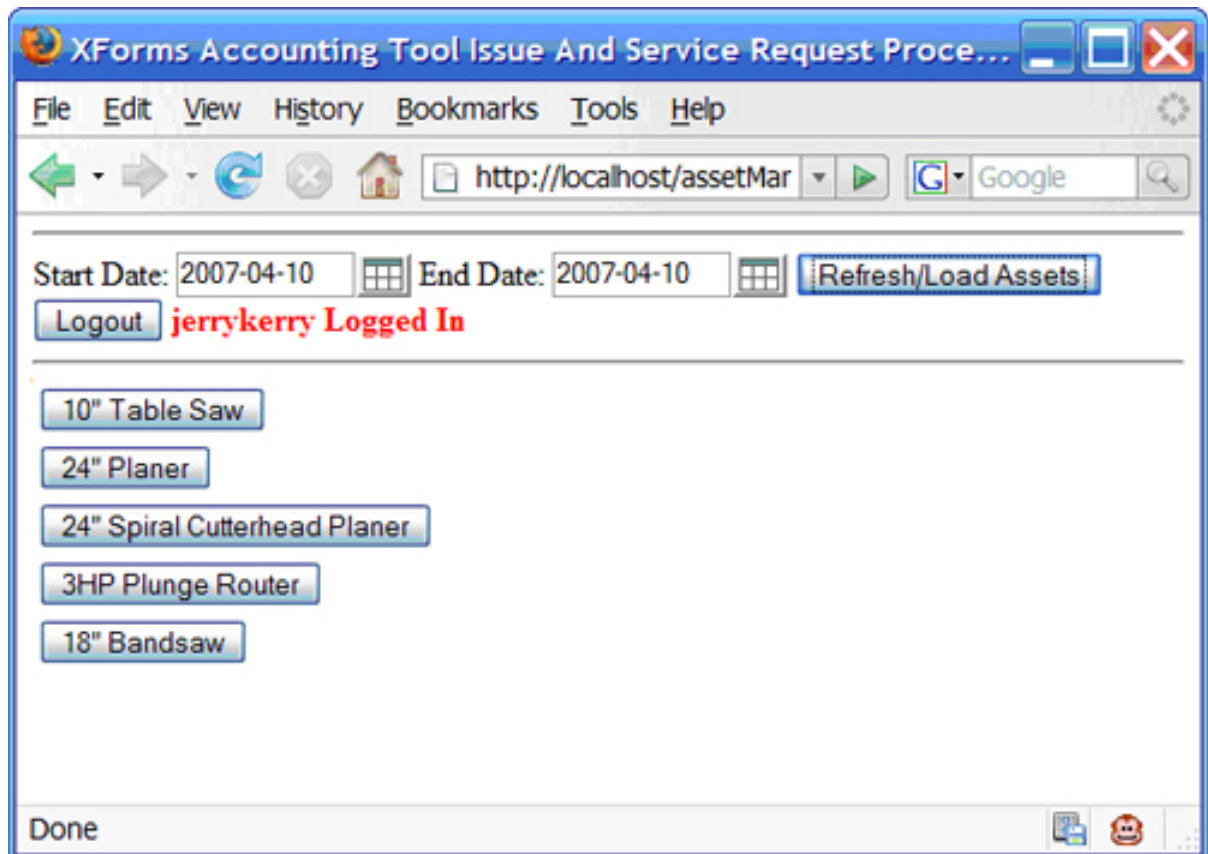
Because we want the user to be able to easily choose an asset for further study, display each asset as a button (see Listing 22).

### Listing 22. Displaying the assets as buttons

```
...
<xforms:output
ref="instance('atuii')/welcome"/>
  <hr />
<xforms:repeat
id="assetSelect"
incremental="true"
nodeset="instance('queryData')/assets/asset">
  <table><tr><td>
<xforms:trigger>
<xforms:label
ref="description"/>
</xforms:trigger>
</td></tr></table>
</xforms:repeat>
  </body>
</html>
```

The result looks like Figure 6.

### Figure 6. The assets as buttons



## Display asset information

When the user clicks one of the buttons, you'll want to display the asset information (see Listing 23).

### Listing 23. Display the asset information

```

...
instance="queryData"
ref="instance('atuii')"
/>

<xforms:bind
nodeset="instance('queryData')//vendor/brief"
calculate="concat(if(..company
= '',
concat(..lastName, ',
',
../firstName),
../company),';
',../street,';
',../city, ',
', ../state, '
', ../zip) " />

<xforms:bind
nodeset="instance('isol')/queryStartDate"

```

```

type="xsd:date" />
...
<xforms:repeat
id="assetSelect"
incremental="true"
nodeset="instance('queryData')/assets/asset">
<table><tr><td>
<xforms:trigger>
<xforms:label
ref="description"/>
<xforms:toggle
ev:event="DOMActivate"
case="browse_issues"/>
</xforms:trigger>
</td></tr></table>
</xforms:repeat>

        <hr />

<xforms:switch>
<xforms:case
id="blank"
selected="true()" />
<xforms:case
id="browse_issues">
<table border="1"
width="100%">
<tr>
<th width="5%"
>Id</th>
<th width="20%"
>Description</th>
<th width="15%"
>Purchase
Date</th>
<th width="10%"
>Status</th>
<th width="50%"
>Vendor</th>
</tr>
<tr>
<td><xforms:output
value=
"instance('queryData')/assets/asset[index('assetSelect')]/assetId"
/></td>
<td><xforms:output
value=
"instance('queryData')/assets/asset[index('assetSelect')]/description"
/></td>
<td><xforms:output
value=
"instance('queryData')/assets/asset[index('assetSelect')]/purchaseDate"
/></td>
<td><xforms:output
value="
if
(instance('queryData')/assets/asset[
index('assetSelect')]/status='retired',
concat('Retired
',
instance('queryData')/assets/asset[index('assetSelect')]/retireDate),
'Active')"/></td>
<td><xforms:output
value=
"instance('queryData')/vendors/vendor[
acctId =
instance('queryData')/assets/asset[index('assetSelect')
]/vendorId]/brief"/>

```

```
</td>
</tr>
</table>

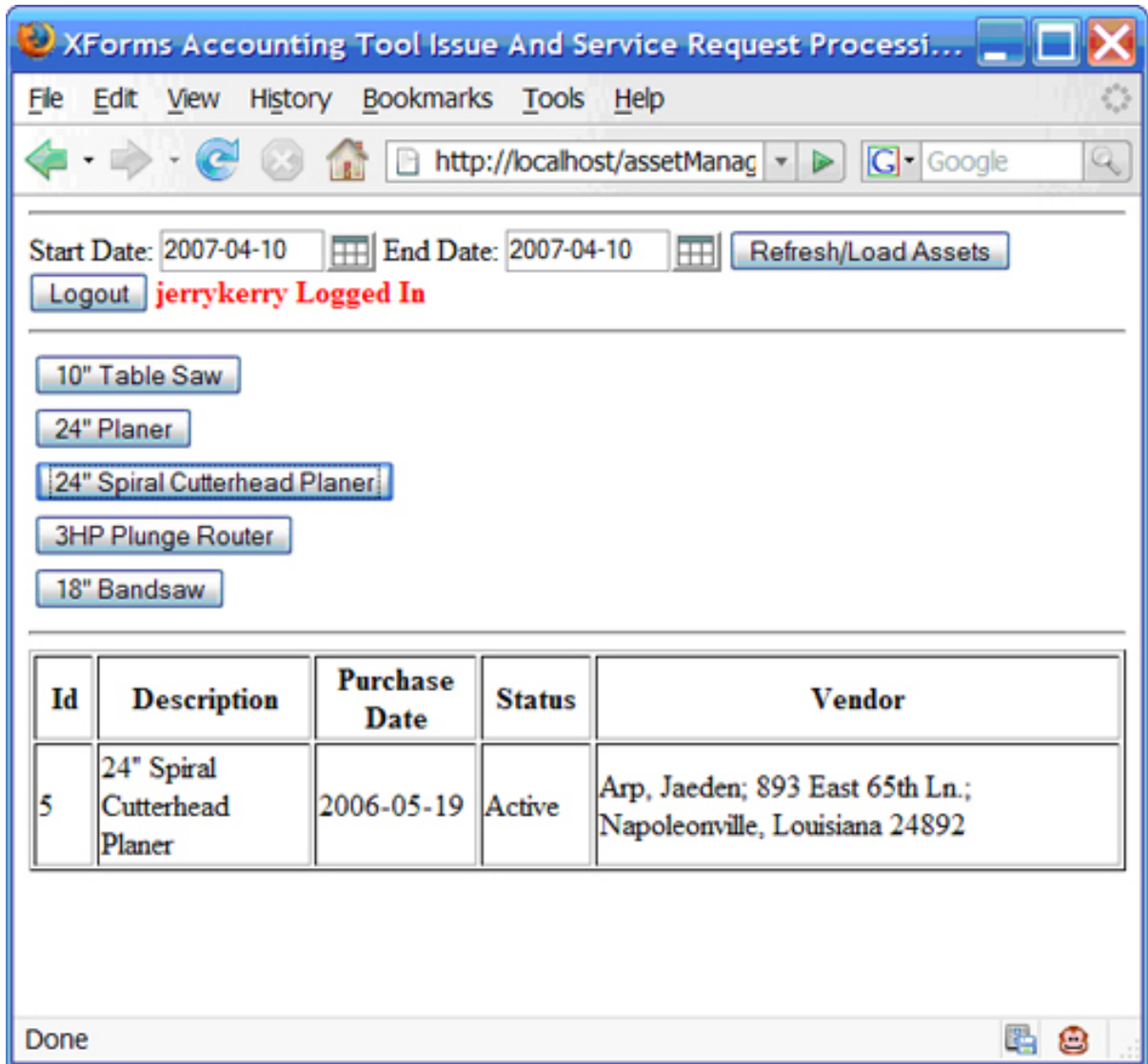
</xforms:case>
</xforms:switch>

  </body>
</html>
```

**Note:** Due to limitations of the Web display, we had to break up some of the XML badly; make sure to put it back together on your own form so it will work.

The day is enclosed in a switch/case statement so that the structural leaps years when appropriate. The toggle in the trigger make sure it's visible. The data itself is fairly straightforward. The interesting part here is the vendor information, which is told not from the asset information, but rather from the vendor section of the instance based on the vendorId, as you can see in Figure 7.

### Figure 7. The asset information



## Display issue information

Each asset and zero or more issues will be displayed (see Listing 24).

### Listings 24. Display the issues

```

...
    acctId =
instance('queryData')/assets/asset[index('assetSelect'
]/vendorId]/brief"/>
</td>
</tr>
<tr>
<td colspan="5">
<table
width="100%">

```

```

<tr>
<th
width="40%">Issue
Reason</th>
<th width="10%">
Requested Action
</th>
<th width="10%">
Issue Request
Date
</th>
<th width="10%">
Current Issue
Status
</th>
<th
width="30%">Issue
Response</th>
</tr>
</table>
</td>
</tr>
<tr>
<td colspan="5">
<xforms:repeat
id="issuesRpt"
nodeset=
"instance('queryData')/assets/asset[index('assetSelect')]/issue">
<table
width="100%">
<tr>
<td width="40%"
align="left">
<xforms:output
ref="reason"/>
</td>
<td width="10%"
align="left">
<xforms:output
ref="action"/>
</td>
<td width="10%"
align="left">
<xforms:output
ref="date"/>
</td>
<td width="10%"
align="left">
<xforms:output
ref="status"/>
</td>
<td width="30%"
align="left">
<xforms:output
ref="response"/>
</td>
</tr>
</table>
</xforms:repeat>
</td>
</tr>
</table>

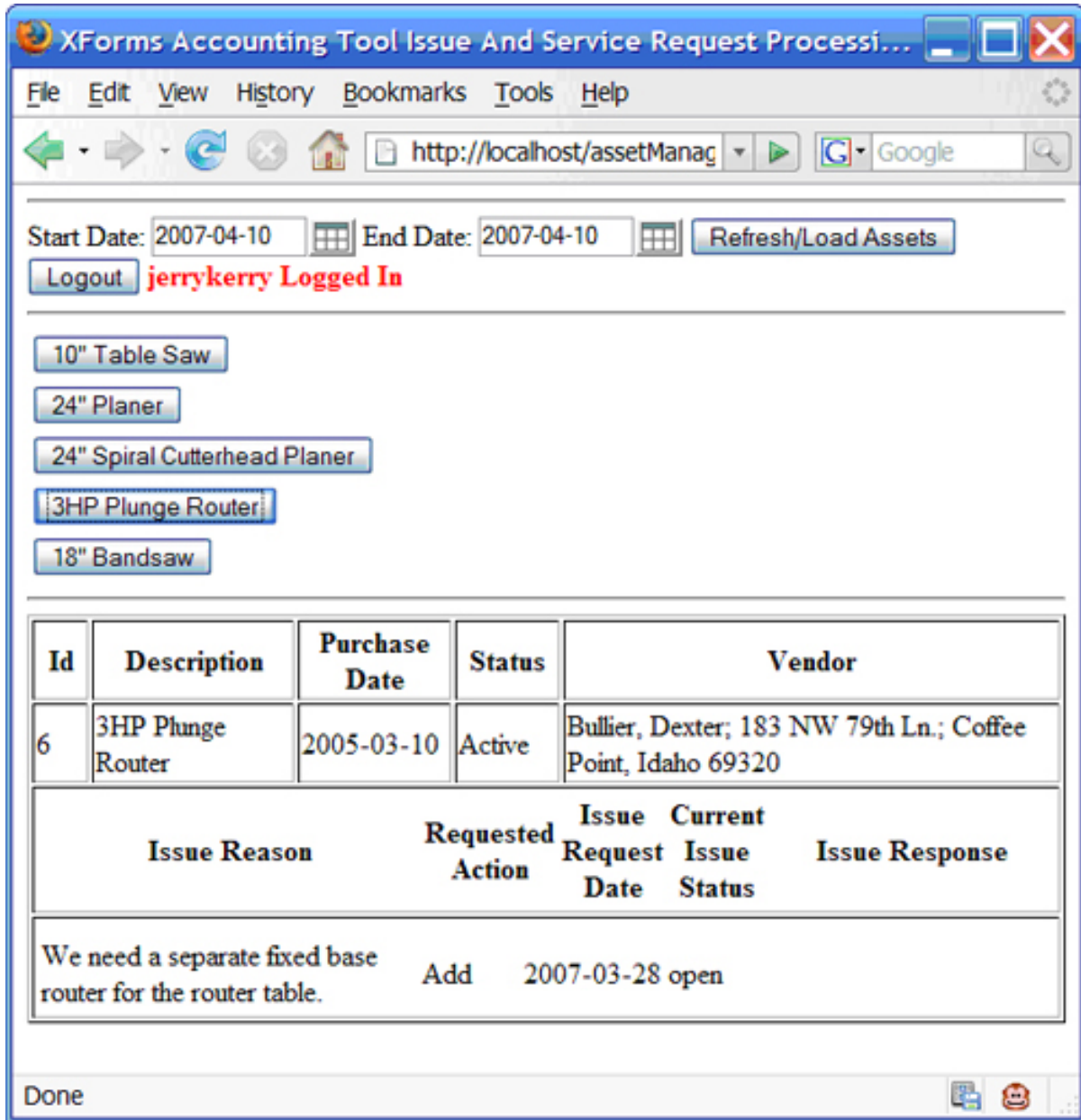
</xforms:case>
</xforms:switch>

</body>
</html>

```

The result looks like Figure 8.

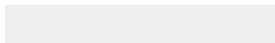
**Figure 8. Display the issues**



### The new issue form

The user of course needs to add new issues, so add before to allow it (the Listing 25).

### Listing 25. The new issue form



```
...
</xforms:repeat>
</td>
</tr>
</table>
<xforms:textarea
ref="instance('atuii')/issueReason">
<xforms:label>
Description Of
Problem:
</xforms:label >
</xforms:textarea>

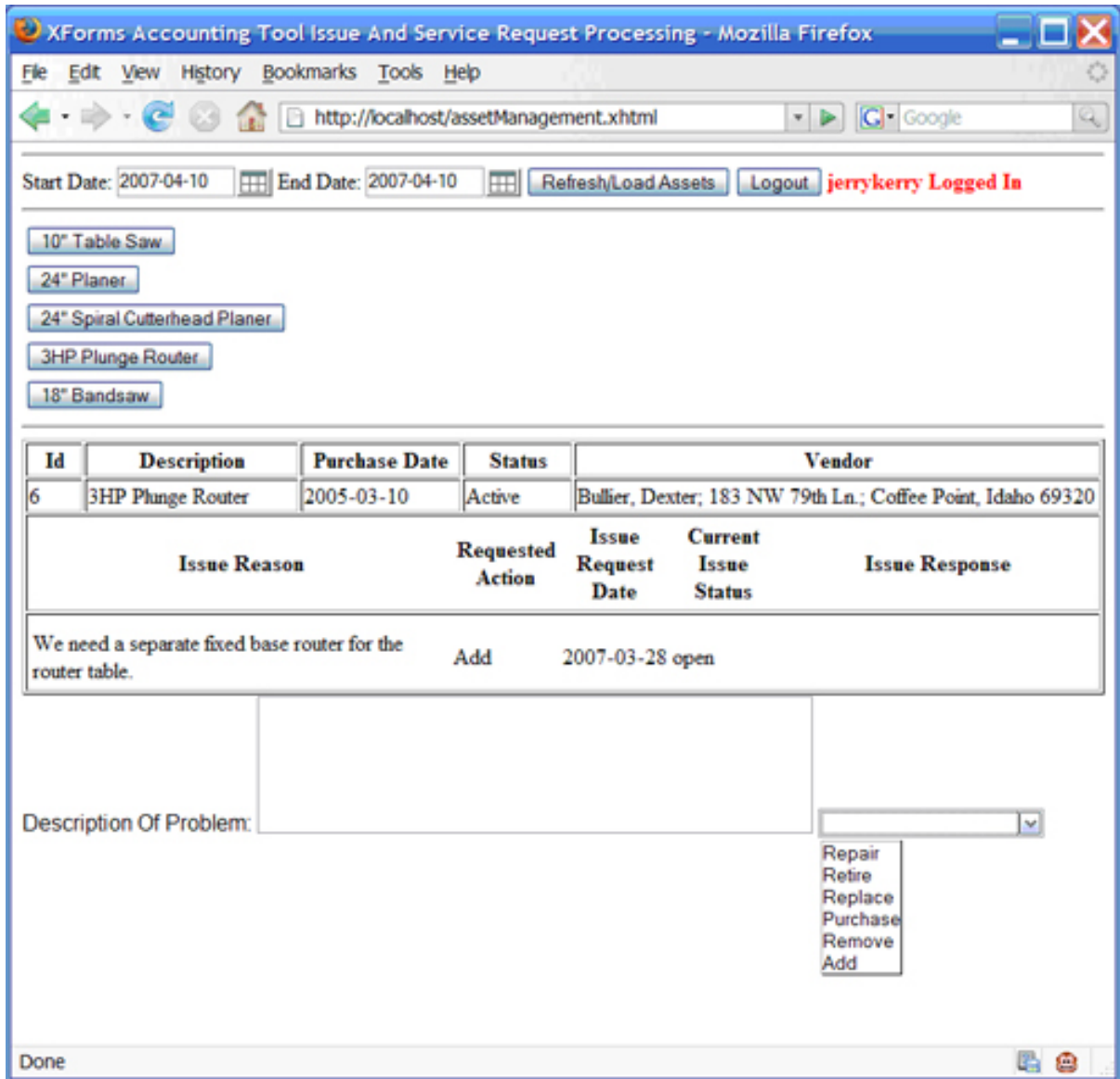
<xforms:select1
ref="instance('atuii')/issueAction">
<xforms:itemset
nodeset="instance('atuii')//issueActionOpts">
<xforms:label
ref="@txt" />
<xforms:value
ref="@txt" />
</xforms:itemset>
</xforms:select1>

</xforms:case>
</xforms:switch>

</body>
</html>
```

Here you are building a select list using the `issueActionOpts` values in the `atuii` instance. The result looks like Figure 9.

### Figure 9. The add issue form



## Starting issue submission

Submitting the form is more than just a matter of calling the PHP script (see Listing 26).

### Listing 26. Submitting the form

```

...
instance="atuii"
ref="instance('atuii')"
/>

<xforms:submission
id="submit_issue"

```

```

action="submit_issue.php"
method="post"
ref="instance('atuii')"
/>

<xforms:submission
id="load_assets"
action="get_assets.php"
...
<xforms:value
ref="@txt"/>
</xforms:itemset>
</xforms:select1>

<xforms:trigger
ref="instance('atuii')/submitIssue">
<xforms:label>Submit</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance('atuii')/issueDate"
value="substring(now(),1,10)"/>
<xforms:setvalue
ref="instance('atuii')/issueStatus"
value=" 'open' "/>
<xforms:setvalue
ref="instance('atuii')/issueAssetId"
value=
"instance('queryData')/assets/asset[index('assetSelect')]/assetId"
/>
<xforms:dispatch
name="xforms-submit"
target="submit_issue"/>
</xforms:action>
</xforms:trigger>

</xforms:case>
</xforms:switch>

</body>
</html>

```

When the user clicks the **Submit** button, the first sets default bodies, such as the `issueDate`, `issueStatus`, and `assetId`. It then manually dispatches the `submit_issue` submission, which calls `submit_issue.php`. The script simply extracts the information from the instance and drops it into the `assetissues` table. (See the [source code](#) for this file.)

## Completing the submission

Once the data has been submitted, you'll need to clean up the form (see Listing 27).

### Listing 27. Clean up the form

```

...
<xforms:dispatch
name="xforms-submit"
target="submit_issue"/>
<xforms:insert

```

```

nodeset=
"instance('queryData')/assets/asset[index('assetSelect')]/issue"
at="1"
position="before"/>

<xforms:setvalue
ref=
"
instance('queryData')/assets/asset[index('assetSelect')]/issue/status
"
value="instance('atuii')/issueStatus"/>
<xforms:setvalue
ref=
"
instance('queryData')/assets/asset[index('assetSelect')]/issue/reason
"
value="instance('atuii')/issueReason"/>
<xforms:setvalue
ref=
"
instance('queryData')/assets/asset[index('assetSelect')]/issue/action
"
value="instance('atuii')/issueAction"/>
<xforms:setvalue
ref=
"
instance('queryData')/assets/asset[index('assetSelect')]/issue/date
"
value="substring(now(),1,10)" />
<xforms:setvalue
ref="instance('atuii')/issueReason"
value=" " />
<xforms:setvalue
ref="instance('atuii')/action"
value=" " />
</xforms:action>
</xforms:trigger>

</xforms:case>
</xforms:switch>

</body>
</html>

```

This is basically a process of moving the data from the form into the live data instance, queryData, and then clearing the form.

---

## Section 7. Procurement users

At the moment, you have a perfectly serviceable asset-management form, enabling users to add issues to existing assets. But someone has to resolve those issues, and that is the responsibility of the procurement department. Users who are part of the procurement department (in this case, department number 123), should have the additional ability to request the "issue view," which lists all of the issues that enables them to add responses.

## Checking for procurement users

The first step is to check to see whether the user is part of the procurement department, and if so, provide the user with a button (see Listing 28).

### Listing 28. Checking for procurement users

```

...
calculate="concat('Welcome
',
instance('atuii')/username)"/>

<xforms:bind
nodeset="instance('atuii')/issueView"
relevant="instance('atuii')/deptId
= '123'"/>

</xforms:model>
...
<xforms:output
ref="instance('atuii')/welcome"/>
  <hr />

<xforms:switch>

<xforms:case
id="standard_view"
selected="true(">

<xforms:trigger
ref="instance('atuii')/issueView">
<xforms:label>Issue
View</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance('atuii')/issueView"
value="1"/>
<xforms:toggle
case="issue_view"/>
</xforms:action>
</xforms:trigger>

      <br/>
      <hr/>

<xforms:repeat
id="assetSelect"
incremental="true"
nodeset="instance('queryData')/assets/asset">
...

</xforms:case>
</xforms:switch>

</xforms:case>
<xforms:case
id="issue_view">

<xforms:trigger
ref="instance('atuii')/issueView">
<xforms:label>Asset
View</xforms:label>

```

```

<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance('atuii')/issueView"
value="0"/>
<xforms:toggle
case="standard_view"/>
</xforms:action>
</xforms:trigger>

</xforms:case>
</xforms:switch>

</body>
</html>

```

Notice that the `issueNew` element is only relevant for users in department 123. They're the only ones who should see the Issue View button. (**Note:** Users are reporting that this button does not appear even for the appropriate department. This appears to be a bug in the Firefox XForms implementation regarding relevance after submissions.)

More than that, however, you now have a nested switch/case scenario. To start with, both the `standard_view` and `blank` cases are selected. When the user clicks an asset button, it activates the `browse_issues` case for the inner switch/case. This has no effect on the outer switch/case. You can manage that switch/case independently using the Issued View and Asset View buttons, which affect only the cases they reference.

## Displaying the issues

Displaying the existing issues is straightforward (see Listing 29).

### Listing 29. Display the issues

```

...
</xforms:trigger>

        <br/>
        <hr/>

<table
width="100%">
<tr>
<th width="5%"
>Asset ID</th>
<th
width="5%">Concerned
Party </th>
<th
width="75%">Issue
Reason</th>
<th
width="5%">Requested
Action </th>
<th

```

```
width="5%">Issue
Request Date
</th>
<th
width="5%">Current
Issue Status</th>
</tr>
</table>

<xforms:repeat
id="issueRpt"
nodeset="instance('queryData')//issue[reason
!= '']">
<table
width="100%">
<tr>
<td align="left"
width="5%" >
<xforms:output
value="../assetId"/>
</td>
<td align="left"
width="5%">
<xforms:output
value="acctId"/>
</td>
<td align="left"
width="75%">
<xforms:output
value="reason"/>
</td>
<td align="left"
width="5%">
<xforms:output
value="action"/>
</td>
<td align="left"
width="5%">
<xforms:output
value="date"/>
</td>
<td align="left"
width="5%">
<xforms:output
value="status"/>
</td>
</tr>
</table>

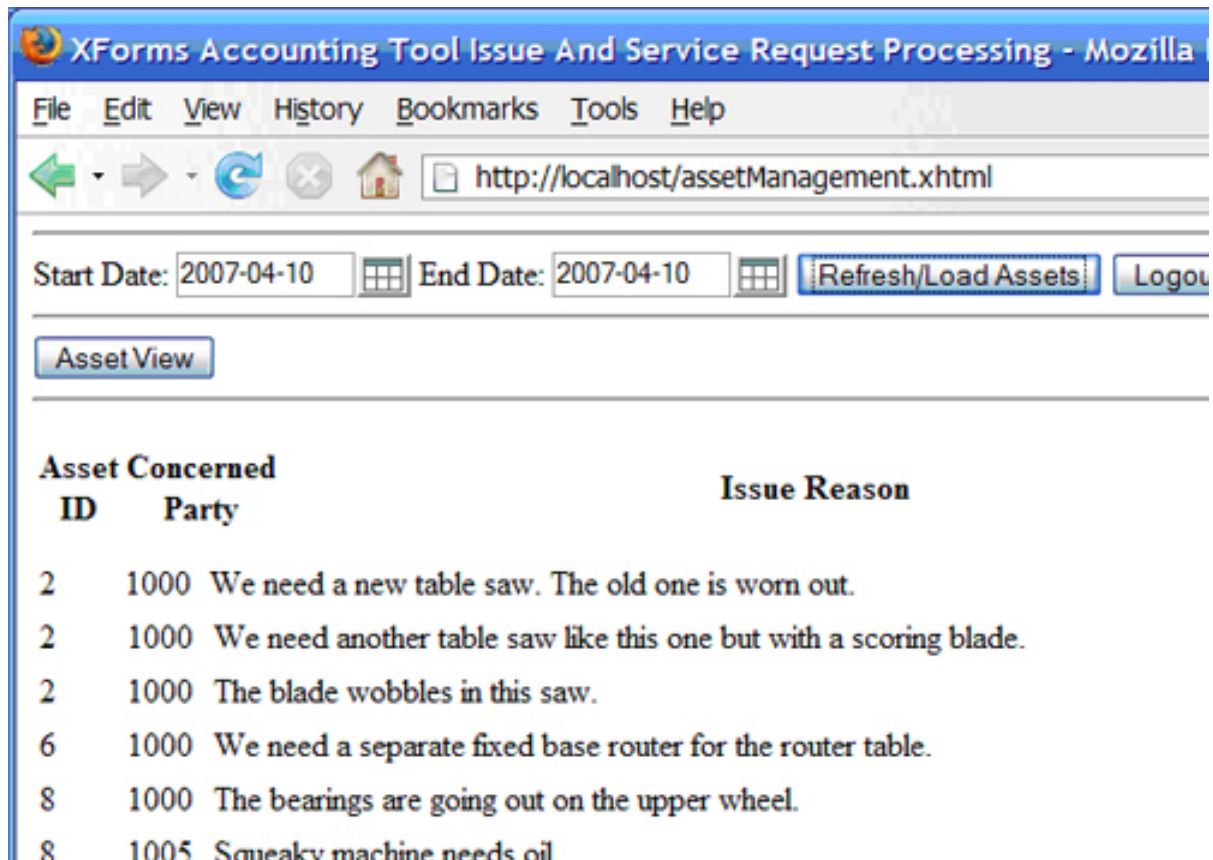
</xforms:repeat>

</xforms:case>
</xforms:switch>

</body>
</html>
```

You can see the results in Figure 10.

### Figure 10. The results



## The one-at-a-time checkbox

Users need a way to select a specific issue, and the checkbox seems like a good idea, but it carries a special challenge (see Listing 30).

### Listing 30. The select box

```

...
<xforms:repeat
id="issueRpt"
nodeset="instance('queryData')//issue[reason
!='']">
<table
width="100%">
<tr>
<td align="left"
width="1%" >
<xforms:select
incremental="true()"
appearance="full"
ref="@selected">

<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance('queryData')//issue[@selected=1]/@selected"
value="0" />

```

```
<xforms:setvalue
ref="." value="1"
/>
</xforms:action>

<xforms:item>
<xforms:label
value=" " />
<xforms:value>1</xforms:value>
</xforms:item>

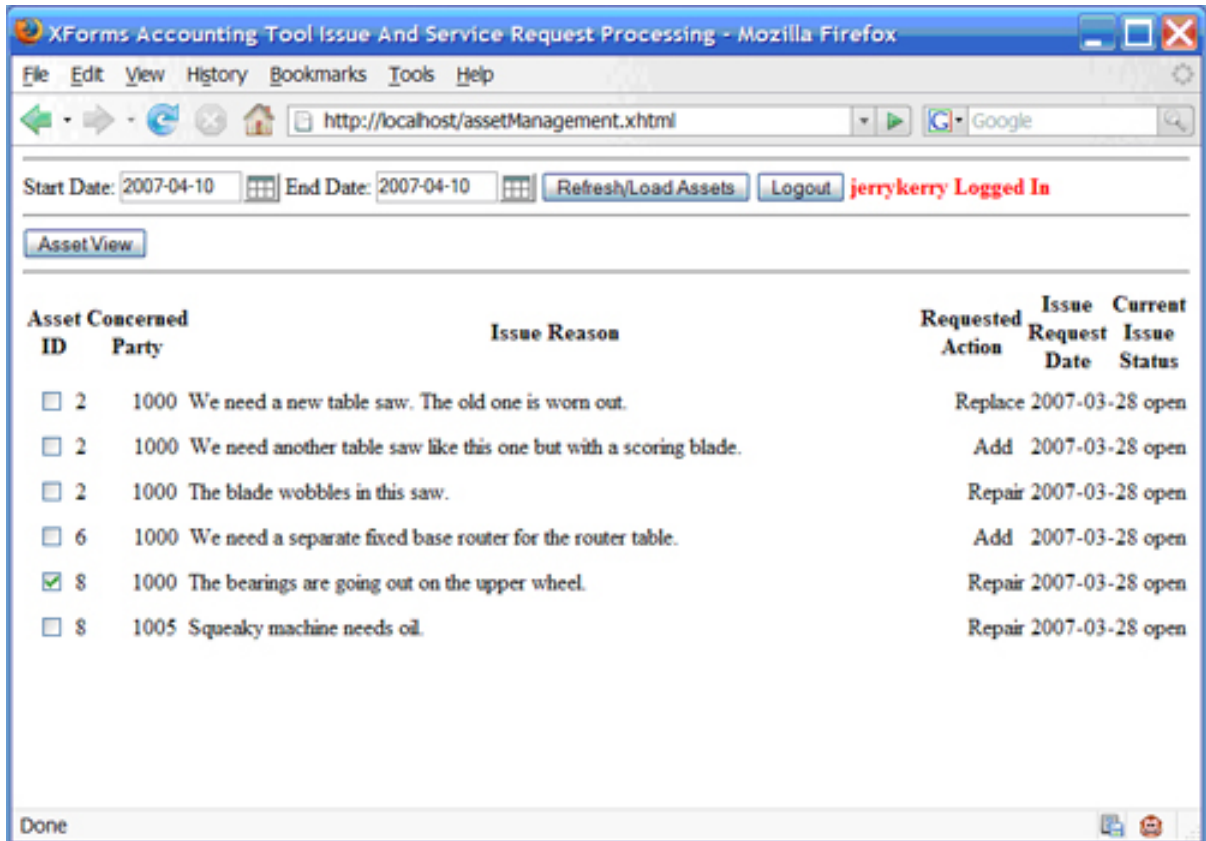
</xforms:select>

</td>
<td align="left"
width="5%" >
<xforms:output
value=" ../assetId" />
</td>
...
```

When you want the user to be able to choose only one option, the `select1` control seems like the perfect choice, but in this case you have a problem. Each iteration of this control is produced by a separate run through the `repeat` element, so each one is an independent item. Rather than having several values of a single `select1` control, you have several `select1` controls, each of which are selectable concurrently.

To solve this problem, you can create a checkbox and manually clear any previously selected items before selecting the current item. This way, when the users select the second checkbox, the first one is cleared and only one gets selected at a time. The result looks like Figure 11.

### Figure 11. The checkbox



## The issue form

Finally, the user needs a way to resolve the issue. You can do that by providing the user with a form (see Listing 31).

### Listing 31. The issue form

```

...
instance="atuii"
ref="instance('atuii')"
/>

<xforms:submission
id="update_issue"
action="update_issue.php"
method="post"
ref="instance('atuii')"
/>

<xforms:submission
id="submit_issue"
action="submit_issue.php"
...
<xforms:bind
nodeset="instance('atuii')/issueView"
relevant="instance('atuii')/deptId
= '123'"/>

```

```

<xforms:bind
nodeset="instance('atuii')/issueRepairInstructions"
calculate=
"instance('queryData')//issue[@selected='1']/reason"/>

<xforms:bind
nodeset="instance('atuii')/password"
relevant="number(instance('atuii')/login)=1"/>
...
<td align="left"
width="5%">
<xforms:output
value="status"/>
</td>
</tr>
</table>

</xforms:repeat>

<xforms:select1
ref="instance('atuii')/issueStatus">
<xforms:label>New
Status:
</xforms:label>
<xforms:itemset
nodeset="instance('atuii')/statusOpts">
<xforms:label
ref="@txt"/>
<xforms:value
ref="@txt"/>
</xforms:itemset>
</xforms:select1>

        <br/>
<xforms:textarea
ref="instance('atuii')/issueResponse">
<xforms:label
>Response:
</xforms:label>
</xforms:textarea>
        <br/>

<xforms:output
ref="instance('atuii')/issueRepairInstructions">
<xforms:label
>Issue Reason:
</xforms:label>
</xforms:output>

        <br/>

<xforms:trigger
ref="instance('atuii')/updateIssue">
<xforms:label>Update
Issue</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:setvalue
ref="instance('atuii')/issueAssetId"
value=
"instance('queryData')//issue[@selected='1']/../assetId"/>
<xforms:dispatch
name="xforms-submit"
target="update_issue"/>
</xforms:action>
</xforms:trigger>

</xforms:case>

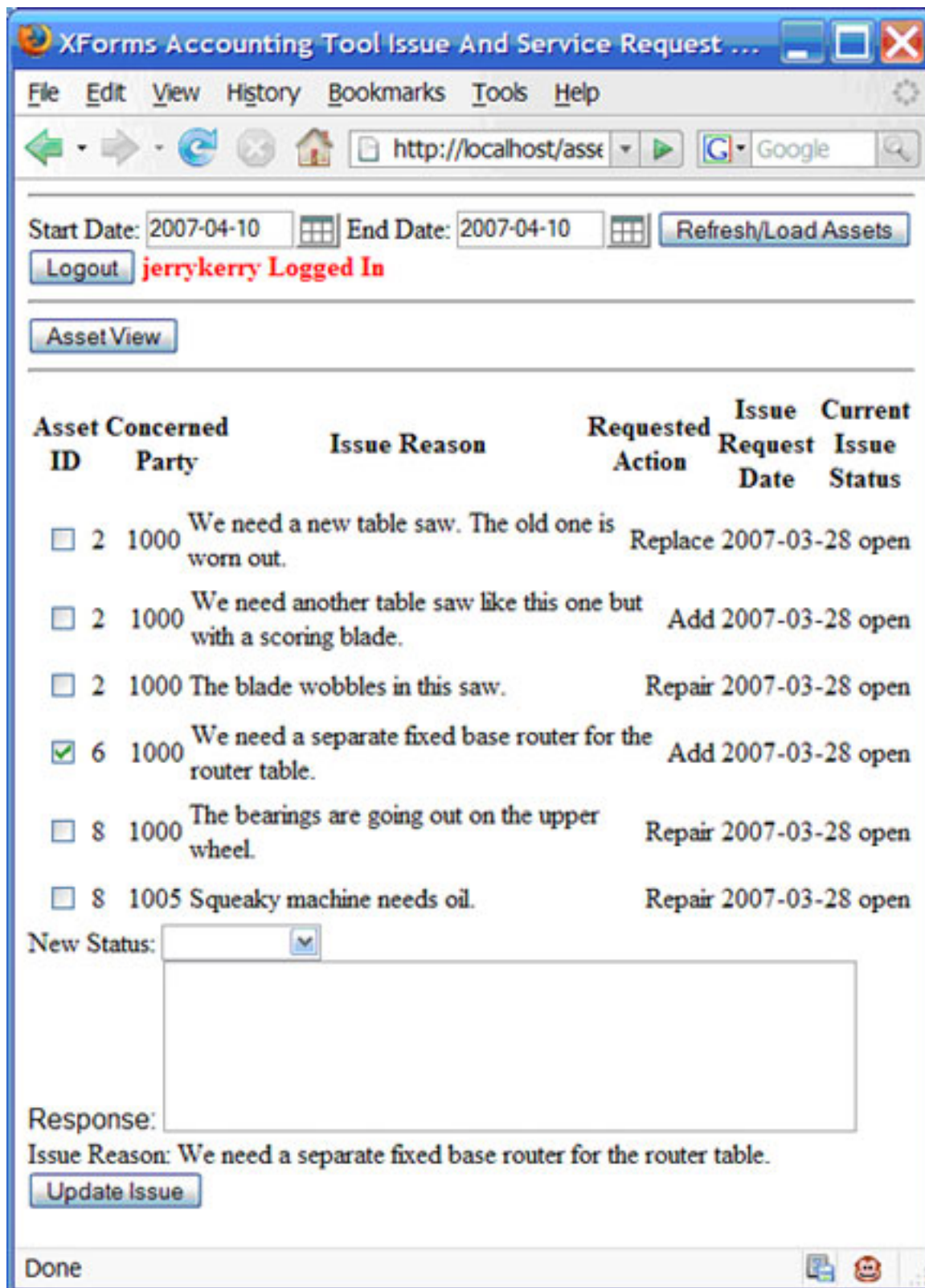
```

```
</xforms:switch>  
  </body>  
</html>
```

The form is pretty simple, providing status options, the text area for a response, and a reiteration of the reason, populated using a bind element. When the user clicks on the **Update Issue** button, it becomes active and the form submits the data to the `update_issue.php` script. That script simply takes the data and drops it in the database. (See the [code download](#) for the `update_issue.php` file.)

The result looks like Figure 12.

### Figure 12. The final form



---

## Section 8. Summary

Tutorials normally are designed to show you one specific skill in isolation, which typically leads to fairly simplistic applications. In this case, however, the purpose is to show you some of the issues you might run into in designing a real-world application such as, in this case, an accounting application.

In this tutorial you learned about issues such as working with multiple database tables and translating them into XML, as well as implementing switch/case statements, and creating a master-detail form.

In Part 5, you'll move on to looking at payments and liabilities for the company and other financial analysis.

## Downloads

Description	Name	Size	Download method
Part 4 source code	xforms4code.zip	11KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- Get a basic introduction to XForms in [Introduction to XForms, Part 1: The new Web standard for forms](#) (developerWorks, September 2006).
- Learn more about SQL injection attacks (and how to prevent them) with [Secure programmer: Call components safely](#) (developerWorks, Decembert 2004).
- Learn more about XForms in the IBM developerWorks [XML zone](#).
- 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.

## 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 authors

## Nicholas Chase

Nicholas Chase has been involved in Web site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. Nick has been a high school physics

teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, an Oracle instructor, and the Chief Technology Officer of an interactive communications company. He is the author of several books, including *XML Primer Plus* (Sams).

---

### 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.