

# Use XForms to create an accounting tool, Part 5: Developing liability management functionality

Skill Level: Intermediate

[Tyler Anderson \(tyleranderson5@yahoo.com\)](mailto:tyleranderson5@yahoo.com)  
Freelance writer

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

08 May 2007

This [six-part series](#) demonstrates how to leverage the power of XForms in conjunction with MySQL and PHP 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 5 of this six-part series demonstrates how to create a payables form for liability and payment, and a reports form to analyze billing data and statistics.

## 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

This tutorial, Part 5 of a [six-part series](#), develops the liability management side of the accounting tool, along with an executive analysis tool that generates reports on the

billing side.

Every business has liabilities and payments, so a tool to help manage and create payments would be beneficial to keep track of things. In this tutorial you'll develop one form, which will be called the payables form, that allows you to create different types of liabilities and payments, including salary, wage and bill types for various departments in the business that can be credited or debited to/from a specified account.

You will also develop a second form, called the reports form, for analyzing billing data and statistics, such as total customer credits and delinquent amounts. You'll also learn how to use the `bind` element and its `calculate` attribute to create a textual representation of a bar graph.

In this tutorial you will learn how to:

- Use `bind` and `calculate` to create two forms
- Select only certain records using XPath in a `repeat` statement
- Use check boxes to select which records to add, save, or remove

## 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 the 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. 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 IBM developerWorks (see [Resources](#)). XForms is built on XML, and, hence, a basic understanding of XML is also assumed.

## 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. Payables: The data

As mentioned in the introduction, you will be creating a payables form and a reports form. In this section you will be focusing on the XML that will be used by the payables form. First you'll take a look at its database schema, then the payables XML structure used by the form, the input XML instance data, and lastly you'll see how to populate the payables XML.

### The database schema revisited

The payables form uses the database to retrieve data regarding various data in a payables record. The [source code](#) for this tutorial has a SQL file that you can upload to your database to create the table shown via SQL in Listing 1.

#### Listing 1. The payables schema

```
DROP TABLE IF
EXISTS
'payables';
CREATE TABLE
'payables' (
'id' int(11)
NOT NULL
auto_increment,
'pmtType'
varchar(20) NOT
```

```

NULL,
  'amt'
decimal(10,0) NOT
NULL,
  'date' date NOT
NULL,
  'dept'
varchar(40) NOT
NULL,
  'desc'
varchar(80) NOT
NULL,
  'gross'
decimal(10,0) NOT
NULL,
  'hours'
decimal(10,0) NOT
NULL,
  'rate'
decimal(10,0) NOT
NULL,
  'recip' int(11)
NOT NULL,
  'recur' int(11)
NOT NULL,
  'recurInc'
int(11) NOT NULL,
  'recurPer'
int(11) NOT NULL,
  PRIMARY KEY
('id')
) ENGINE=InnoDB
DEFAULT
CHARSET=latin1
AUTO_INCREMENT=4
;

```

The `id` is a self identifier, and the `recip` is the `acctid` of the recipient. The rest of the fields are fairly self-explanatory, so let's move to the next section where you'll learn the XML structure.

## The target structure

The payables form has an XML structure that contains a list of departments and payables to populate the form data with (see Listing 2).

### Listing 2. XML structure for the payables instance data

```

<payablesForm>
  <departments>
    <department
name="United
Cabinetry"
real="1"/>
    <department
name="Tables And
Chairs"
real="1"/>
    <department
name="Boxgoods"

```

```

real="1"/>
  <department
name="Blanket
Chests"
real="1"/>
  <department
name="Dressers"
real="1"/>
  <department
name="Flooring"
real="1"/>
  <department
name="Custom
Flooring"
real="1"/>
  <department
name="Plank
Flooring"
real="1"/>
  </departments>
  <payables>
  <payable
pmtType="" amt=""
date="" dept=""
desc="" gross=""
hours="" rate=""
real="0" recip=""
recur=""
recurInc=""
recurPer=""
selected="0"/>
  <payable
id="4"
pmtType="Wage"
amt="120"
date="2007-12-31"
dept="Boxgoods"
desc="New
payment"
gross="0"
hours="0"
rate="0"
recip="5"
recur="1"
recurInc="1"
recurPer="365"
selected="0"
real="1">
  <acct
acctId="5"
firstName="Jaeden"
lastName="Arp"
username="jaedenarp"
password="foo128"
street="893 East
65th Ln."
city="Napoleonville"
state="Louisiana"
country="USA"
zip="24892"
phone="(645)
847-8741"
deptId="0"
company=""
real="1"/>
  </payable>
  <payable
id="6"
pmtType="Wage"

```

```
amt="120"
date="2007-12-31"
dept="Boxgoods"
desc="New
payment3"
gross="0"
hours="0"
rate="0"
recip="3"
recur="1"
recurInc="1"
recurPer="365"
selected="0"
real="1">
  <acct
acctId="3"
firstName="Dexter"
lastName="Bullier"
username="dexterbullier"
password="fool26"
street="183 NW
79th Ln."
city="Coffee
Point"
state="Idaho"
country="USA"
zip="69320"
phone="(109)
611-9170"
deptId="0"
company=""
real="1"/>
  </payable>
</payables>
</payablesForm>
```

In Listing 2 the first set of elements are `departments` elements. Each payment is associated with a department, and users will select which department's account payables they want to view in the form. A payable is also associated with an account, and so you should notice the `<acct . . .>` elements within the `<payable . . .>` elements. The account data is used to display information about the account in connection with the payable on the form.

The first payable's `real` attribute is 0, which means that it will not be shown in the form and it will be used as a template to add new payables within the form.

Next, you'll take a look at the input processing XML structure.

## Processing input

The input XML instance data allows you to have "local variables" within the XForm document that you can manipulate and tie to `select1` and other input elements, which you can then use in `bind` elements for any calculations you might want.

The input XML instance data for the payables form contains various fields related to logging in and other controls and features contained within the form, which you can

see in the instance document (see Listing 3).

### Listing 3. The input XML instance data

```
<xforms:instance
id="atui">
  <atui
xmlns="">
  <loginData>
  <loginLogout>Login</loginLogout>
  <password/>
  <login>1</login>
  <loginToken/>
  <username/>
  <deptId/>
  <welcome/>
  </loginData>
  <errorMessage/>
  <refresh/>
  <debug>1</debug>
  <controls>
  <saveSelected/>
  <newPayable/>
  <removePayable/>
  <filters>
  <pmtType>*</pmtType>
  <dept>*</dept>
  </filters>
  </controls>
    </atui>
  </xforms:instance>
```

The elements under `<loginData>` are related to logging in, and features relating to the form are under `<controls>`. There are two `select1` elements showing a list of payable types and another listing the departments, and the contents of those `select1` elements will be stored in `<pmtType>` and `<dept>`, both of which are under `controls/filters`. Next, take a look at how the payables instance data is populated.

## Populating departments

The departments are used to populate a `select1` element, and you'll see how to populate them for the payables XML instance data in Listing 4.

### Listing 4. Populating department XML data

```
$query =
sprintf('SELECT *
FROM
`departments`');
$departmentRows =
mysql_query($query,
$sqlldb);
if
(!$departmentRows)
die ('Could
```

```

not execute acct
db query:
<b>'. $query.
'</b>
because: ' .
mysql_error();
while
($departmentRow =
mysql_fetch_assoc($departmentRows))
{
    $department =
$departmentRow['name'];
$newDataSrc=$doc->createElement('department');
$newDataSrc->setAttribute('name',sprintf("%s",$department));
$newDataSrc->setAttribute('real','1');
$rootNode->GetElementsByTagName('departments')->
item(0)->appendChild($newDataSrc);
}

```

Notice the data comes from the departments table, and then each department contained is simply stored in a new element and added to the instance document. Next, see how to populate payables.

## Populating payables

When payables records are stored in the database, they can be later retrieved as well, and Listing 5 shows you how to populate the instance data.

### Listing 5. Populating payables records in the XML instance data

```

$query =
sprintf('SELECT *
FROM
'payables');
$payablesRows =
mysql_query($query,
$sqlldb);
if
(!$payablesRows)
    die ('Could
not execute acct
db query:
<b>'. $query.
'</b>
because: ' .
mysql_error());
while
($payablesRow =
mysql_fetch_assoc($payablesRows))
{
    $newDataSrc=$doc->createElement('payable');
    foreach
($payablesRow as
$key => $val) {
        $newDataSrc->setAttribute($key,
        $val);
    }
    $newDataSrc->setAttribute('selected','0');
    $newDataSrc->setAttribute('real','1');
    $rootNode->GetElementsByTagName('payables')->

```

```

item(0)->appendChild($newDataSrc);
...
}

```

Here new `payable` elements are created by adding a new attribute to the new payable record for every field contained in the payables table. Two other custom attributes are also added, `selected` and `real`; `selected=0` indicates that it will not appear selected in the form, and `real=1` indicates that it is not the dummy element and is real data to be used in the form. Let's move to the final step: populating accounts.

## Populating accounts

Lastly, the associated account needs to be added as a child to the `<payable>` element (see Listing 6).

### Listing 6. Populating accounts data

```

...
$newDataSrc->setAttribute('real','1');
$rootNode->GetElementsByTagName('payables')->
item(0)->appendChild($newDataSrc);

    $query =
sprintf('SELECT *
FROM 'accounts'
WHERE ' .
'acctId'=%d;', $newDataSrc->GetAttribute('recip'));
    $accountsRows
=
mysql_query($query,
$sqlldb);
    if
(! $accountsRows)
die ('Could not
execute acct db
query: <b>' .
$query. '</b>
because: ' .
mysql_error());
    while
($accountsRow =
mysql_fetch_assoc($accountsRows))
{
    $newDataSrc2=$doc->createElement('acct');
    foreach
($accountsRow as
$key => $val) {
    $newDataSrc2->setAttribute($key,
$val);
    }
    $newDataSrc2->setAttribute('real','1');
    $newDataSrc->appendChild($newDataSrc2);
}

```

The above, in Listing 6, is similar to how you populated payables data in [Listing 5](#),

except you search for the account whose ID matches the `acctid` contained in `recip`. The resulting row(s) from the SQL query are then added as a child to the `payable` element, ready for your use later in the form.

---

## Section 3. Payables: The form

Now on to the good stuff! In this section you'll take the XML documents you just learned about and created, and feed them into the payables form that will display the data appropriately.

### The essentials

First you'll start off with the form's basics shown in Listing 7. This is where the users of X-Trapolate need to login to the payable form.

#### Listing 7. Basics of the payables form

```
<!DOCTYPE html
PUBLIC
"-//W3C//DTD
XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.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"
>
  <head>
    <title>Accounting
    Tool
    Payables</title>

    <xforms:model
    id="atxm">

    <xforms:instance
    id="payables">
      <payablesForm
      xmlns="">
      </payablesForm>
    </xforms:instance>

    <xforms:instance
    id="atui">
      <atui
      xmlns="">
      <loginData>
      <loginLogout>Login</loginLogout>
      <password/>
      <login>l</login>
      <loginToken/>
```

```

<username/>
<deptId/>
<welcome/>
</loginData>
<errorMessage/>
<refresh/>
<debug>1</debug>
<controls>
<saveSelected/>
<newPayable/>
<removePayable/>
<filters>
<pmtType>*</pmtType>
<dept>*</dept>
</filters>
</controls>
    </atui>
</xforms:instance>

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

<xforms:submission
id="login_logout"
action="login_logout.php"
method="post"
replace="instance"
instance="atui"
ref="instance('atui')"
/>

<xforms:action
ev:event="xforms-ready">
<xforms:dispatch
name="xforms-submit"
target="login_logout"/>
<xforms:dispatch
name="xforms-submit"
target="load_payables"/>
</xforms:action>

</xforms:model>
<style>
*.required:after
{ content: "*";
color: red; }
*.required
{
font-weight:bold;
}
*.errorMessage {
font-weight:
bold; color: red;
}
</style>
</head>
<body >
<xforms:input

```

```

ref="instance('atui')//username">
<xforms:label>Username:
</xforms:label></xforms:input>
<xforms:secret
ref="instance('atui')//password">
<xforms:label>Password:
</xforms:label></xforms:secret>
<xforms:submit
submission="login_logout"
ref="instance('atui')//loginLogout">
<xforms:label
ref="instance('atui')//loginLogout"/></xforms:submit>
<xforms:output
ref="instance('atui')//welcome"/><br/>
<xforms:output
class="errorMessage"
value="concat(instance('atui')//errorMessage,
instance('queryData')/errorMessage)"/>
...

```

Listing 7 shows how the input instance data fits in with the rest of the form, and also the login portion of the form, as used in previous parts of this series. Note the `action` element, which validates the user's login information each time a logged in user visits the form. The `load_payables` target is then called, which immediately loads the form with instance data that you can then view in the form. The next few sections will add the form body.

## Adding payables selection criteria

To get anything useful out of the form, you'll need to be able to single out payables such as the Boxgoods department and Wage payment type. Remember the `select1` elements that have been talked about? You'll add them in Listing 8.

### Listing 8. Selecting payables attributes

```

<table
width="100%"
border="1px"
height="100%">
  <tr
height="5%">
    <td
width="10%"
rowspan="2"
valign="top">
<xforms:select1
ref="instance('atui')/controls/filters/pmtType"
incremental="true()">
<xforms:label>Payment
Type: <br/></xforms:label>
<xforms:item>
<xforms:label>Salary</xforms:label>
<xforms:value>Salary</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Wage</xforms:label>
<xforms:value>Wage</xforms:value>
</xforms:item>

```

```

<xforms:item>
<xforms:label>Bill</xforms:label>
<xforms:value>Bill</xforms:value>
</xforms:item>
</xforms:select1>
  <hr/>
<xforms:select1
ref="instance('atui')/controls/filters/dept"
incremental="true()">
<xforms:label>Department:<br/></xforms:label>
<xforms:itemset
nodeset="instance('payables')/departments//department">
<xforms:label
ref="@name"/>
<xforms:value
ref="@name"/>
</xforms:itemset>
</xforms:select1>
  </td>
  <td
width="90%">
<xforms:trigger
ref="instance('atui')/controls/saveSelected"
submission="add_payables">
<xforms:label>Save/Add
Selected
Payments</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:dispatch
name="xforms-submit"
target="add_payables"/>
<xforms:dispatch
name="xforms-submit"
target="load_payables"/>
</xforms:action>
</xforms:trigger>
<xforms:trigger
ref="instance('atui')/controls/newPayable">
<xforms:label>Create
New
Payment</xforms:label>
<xforms:action
ev:event="DOMActivate">
<xforms:insert
nodeset="instance('payables')//payable"
at="count(instance('payables')//payable)-1"
position="after"/>
<xforms:setvalue
ref="instance('payables')//payable[count(instance('payables')//payable)]/@real"
value="1"/>
<xforms:setvalue
ref="instance('payables')//payable[count(instance('payables')//payable)]/@id"
value="'-1'"/>
<xforms:setvalue
ref="instance('payables')//payable[count(instance('payables')//payable)]/@pmtType"
value="instance('atui')/controls/filters/pmtType"/>
<xforms:setvalue
ref="instance('payables')//payable[count(instance('payables')//payable)]/@dept"
value="instance('atui')/controls/filters/dept"/>

</xforms:action>
</xforms:trigger>
<xforms:submit
ref="instance('atui')/controls/removePayable"
submission="remove_payables">
<xforms:label>Remove
Selected
Payments</xforms:label>

```

```
</xforms:submit>
  </td>
</tr>
```

You'll see how the first two `select1` elements are set up. The first one contains hard-coded items: `*`, `Wage`, `Salary` and `Bill` as elements. The second one retrieves its data from the departments listed in the payables instance data. The next few elements are a couple of triggers and a submission element. Notice the "Save/Add Selected Payments" trigger. This trigger sends the current payables instance data over to a PHP file that saves changes and inserts new additions, as specified by check boxes. Changes are then reloaded back into the form. The "Create New Payment" trigger is a little more involved in that it adds a new payable element to the form, and then sets its ID to `-1` (having this new payable record saved in the database sets its ID), its department to the currently selected department in the `select1` element, and also its payment type to the currently selected payment type in the other `select1` element. Its `real` attribute is also set to `1`. The last button, a submission, sends the instance data to another PHP file that removes selected elements from the database. You learn more about the two PHP files mentioned later in this section.

The submission elements that [Listing 8](#) above triggers are shown in [Listing 9](#).

### Listing 9. Submission elements

```
<xforms:submission
id="remove_payables"
action="remove_payable.php"
method="post"
replace="instance"
instance="payables"
ref="instance('payables')"/>

<xforms:submission
id="add_payables"
action="add_payable.php"
method="post"
ref="instance('payables')"/>

<xforms:submission
id="load_payables"
action="load_payables.php"
method="post"
replace="instance"
instance="payables"
ref="instance('atui')"/>
```

Note that the `add_payables` submission element only sends the instance data. The refreshed data is retrieved using the `load_payables` submission element.

Next you'll move onto the rest of the form.

## Viewing payables

Now you need to provide the ability for X-Trapolate users to view the payables they have selected. You'll do so using a `repeat` element that displays payable data. Take a look at it first in Listing 10.

### Listing 10. Viewing payables

```

        <tr>
        <td
width="90%">
<xforms:repeat
id="payRpt"
nodeset="instance('payables')//payable[@real='1'][@pmtType=instance('atui')
/controls/filters/pmtType][@dept=instance('atui')/controls/filters/dept]">
<table
border="1px"
width="100%">
<tr>
<td rowspan="3"
width="5%">
<xforms:select
appearance="full"
ref="@selected">
<xforms:item>
<xforms:value>1</xforms:value>
</xforms:item>
</xforms:select>
</td>
</tr>
<tr>
<td width="95%">
<xforms:input
ref="@desc">
<xforms:label>Description:
</xforms:label>
</xforms:input>
<xforms:input
ref="@amt">
<xforms:label>Amount:
</xforms:label>
</xforms:input>
<xforms:input
ref="@date">
<xforms:label>Date
Due:
</xforms:label>
</xforms:input>
<xforms:select1
ref="@recurPer">
<xforms:label>Frequency:</xforms:label>
<xforms:item>
<xforms:value>1</xforms:value>
<xforms:label>Daily</xforms:label>
</xforms:item>
<xforms:item>
<xforms:value>7</xforms:value>
<xforms:label>Weekly</xforms:label>
</xforms:item>
<xforms:item>
<xforms:value>30</xforms:value>
<xforms:label>Monthly</xforms:label>
</xforms:item>

```

```

<xforms:item>
<xforms:value>90</xforms:value>
<xforms:label>Quarterly</xforms:label>
</xforms:item>
<xforms:item>
<xforms:value>365</xforms:value>
<xforms:label>Yearly</xforms:label>
</xforms:item>
</xforms:select1>
<xforms:input
ref="@recurInc">
<xforms:label>Period:
</xforms:label>
</xforms:input><br/>
<xforms:select
appearance="full"
ref="@recur">
<xforms:item>
<xforms:label>Recurring</xforms:label>
<xforms:value>1</xforms:value>
</xforms:item>
</xforms:select>
<xforms:input
ref="@recip">
<xforms:label>Payee's
account #:
</xforms:label>
</xforms:input><br/>
Payee's
information:
<table
border="1px"><tr><td>
<xforms:output
value="concat(acct/@firstName,
'
',acct/@lastName)"/><br/>
<xforms:output
value="acct/@street"/><br/>
<xforms:output
value="concat(acct/@city,'
',acct/@state,
'
',acct/@zip)"/>
<br/>
</td></tr></table>
</td>
</tr>
</table>
</xforms:repeat>
</td>
</tr>
</table>
</body>
</html>

```

The novel part of this `repeat` statement, perhaps, is its `nodeset` attribute shown in Listing 11.

### Listing 11. nodeset attribute

```

instance('payables')//payable[@real='1'][@pmtType=instance('atui')
/controls/filters/pmtType][@dept=instance('atui')/controls/filters/dept]

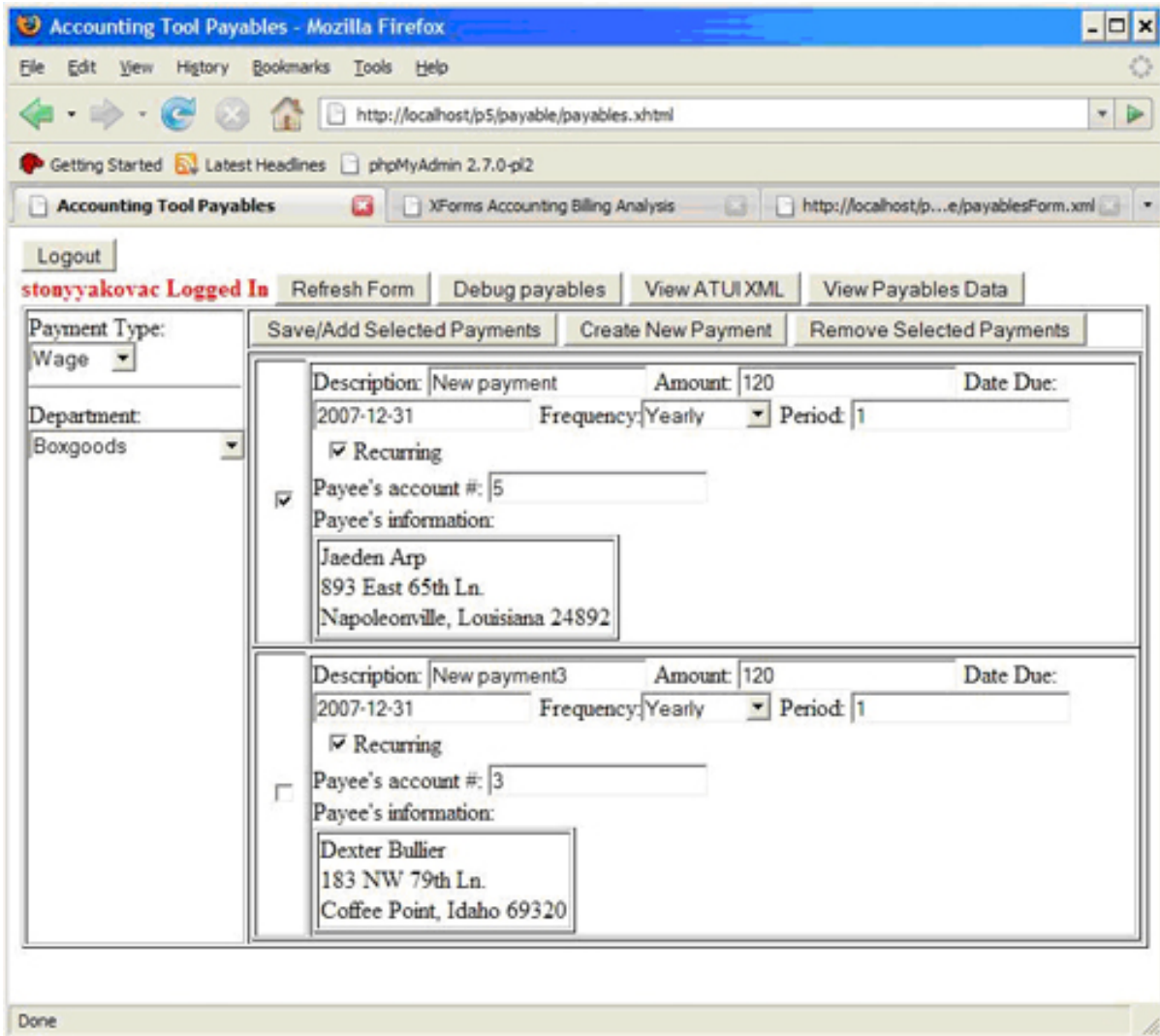
```

It selects all items that are real (`@real='1'`), and whose payment type equals the one currently selected in the `select1` element, indicating the payment type; and lastly it selects the department as specified in the other `select1` element. Thus, only the payables applicable to the payment type and department selected are shown.

Within the `repeat` element is a check box allowing you to "select" each element. Selecting an element allows you to specify whether or not to remove an element from the database or save/add changes. The payable data is then displayed using input elements, a checkbox, and a `select1` element. The information relating to the recipient of the payable is then displayed at the end using output elements. Note that the adding/saving and removing functionality won't work until you define the PHP for it, as explained in the following sections.

The form is shown in Figure 1.

### **Figure 1. The payables form**



It's a pretty slick form! I hope you'll have as much fun playing with it as it was creating it. Next, you'll go over how the PHP handles adding/saving and removing selected payables.

## Adding/Saving selected payables

Once a new payable is added, or an existing one is changed, they need to be selected to be added/saved to the database. The PHP to do it is shown in Listing 12.

### Listing 12. Updating existing and inserting new payable records

```
$payables =
$doc->GetElementsByTagName( "payable" );
for($i=0;$i <
$payables->length;$i++)
{
```

```

        if
($payables->item($i)->GetAttribute('selected')
== 1) {
    $payable =
$payables->item($i);

    $id =
$payable->GetAttribute('id');
    if($id >
0){ // update
        $query
= sprintf("UPDATE
'payables' set
'pmtType' = '%s',
'amt' = '%s',
'date' = '%s',
'dept' = '%s',
'desc' = '%s',
'gross' = '%s',
'hours' = '%s',
'rate' = '%s',
'recip' = '%s',
'recur' = '%s',
'recurInc' =
's',
'recurPer' = '%s'
where 'id' =
%d;",
$payable->GetAttribute('pmtType'),
$payable->GetAttribute('amt'),
$payable->GetAttribute('date'),
$payable->GetAttribute('dept'),
$payable->GetAttribute('desc'),
$payable->GetAttribute('gross'),
$payable->GetAttribute('hours'),
$payable->GetAttribute('rate'),
$payable->GetAttribute('recip'),
$payable->GetAttribute('recur'),
$payable->GetAttribute('recurInc'),
$payable->GetAttribute('recurPer'),
$payable->GetAttribute('id'));
        $addQuery =
mysql_query($query,
$sqlldb);
        if
(!$addQuery)
            die
('Could not
execute acct db
query:
<b>'. $query.
'</b> because: '
. mysql_error());
        } else { //
insert
            $query
= sprintf("INSERT
INTO 'payables' (
'id' , 'pmtType'
,
'amt' , 'date' ,
'dept' , 'desc' ,
'gross' , 'hours'
, 'rate' ,
'recip' , 'recur'
,
'recurInc' ,
'recurPer' )
VALUES ( NULL ,

```

```

'%s', '%s', '%s',
'%s', '%s', '%s',
'%s',
'%s', '%s', '%s',
'%s', '%s');",
$payable->GetAttribute('pmtType'),
$payable->GetAttribute('amt'),
$payable->GetAttribute('date'),
$payable->GetAttribute('dept'),
$payable->GetAttribute('desc'),
$payable->GetAttribute('gross'),
$payable->GetAttribute('hours'),
$payable->GetAttribute('rate'),
$payable->GetAttribute('recip'),
$payable->GetAttribute('recur'),
$payable->GetAttribute('recurInc'),
$payable->GetAttribute('recurPer'));
$addQuery =
mysql_query($query,
$sqlldb);
        if
(! $addQuery)
            die
('Could not
execute acct db
query:
<b>'. $query.
'</b> because: '
. mysql_error());
    }
}
}

```

See how all elements are iterated, and if `selected=1`, meaning the checkbox is checked, then they'll be considered for saving/adding to the database. If the `id=-1` then the payable record is a new record and it needs to be inserted using `INSERT` rather than `UPDATE`. Otherwise, the record is updated in the database using `UPDATE`.

Next, you'll see how to remove elements in a similar fashion.

## Removing selected payables

When items are selected for removal, the instance data ends up in the PHP shown in Listing 13.

### Listing 13. Removing selected payables

```

$payables =
$doc->GetElementsByTagName("payable");
for($i=0;$i <
$payables->length;$i++)
{
    if
($payables->item($i)->GetAttribute('selected')
== 1) {
        $query =
sprintf('DELETE

```

```

FROM 'payables'
WHERE
'payables'.'id'
'.'
      '=
%d LIMIT
1;', $payables->item($i)->GetAttribute('id'));
$removeQuery =
mysql_query($query,
$sqlldb);
      if
(! $removeQuery)
        die
('Could not
execute acct db
query:
<b>' . $query .
'</b> because: '
. mysql_error());
    }
}
for($i=0;$i <
$payables->length;$i++)
{
    if
($payables->item($i)->GetAttribute('selected')
== 1) {
$doc->GetElementsByTagName("payables")->
item(0)->removeChild($payables->item($i));
    }
}
echo
$doc->saveXML();

```

Similar to adding/saving payables records, here you iterate over all the payables and if one was selected for removal, it is removed from the database. The second `for` loop then removes all payables selected from the instance document, which is then sent back to the form where removed elements will no longer be displayed.

That finishes off the payables form. Next, you'll develop the reports form that will allow you to analyze billing data (from the billing form in [Part 3](#) of this series).

## Section 4. Reports form: The data

The reports form, which you'll develop in this and the next section, allows you to view totals and statistical information (shown in a bar graph) about billing data for the company. First, let's go over the XML data used by the form in this section.

### The target structure

The XML instance data used by this form comes from the same data used for the

billing form in [Part 3](#) of this tutorial series. This form essentially creates an analysis tool on the billing data that allows you to see a graphical representation of what state accounts are in (past due, current, etc.), and to also view totals such as total customer credits (customers the company owes money to). Note that the data could also come from another database, and you can create additional reports based on the data being analyzed. Check out the associated instance data in Listing 14.

### Listing 14. Analyze XML instance data

```
<?xml
version="1.0"
encoding="UTF-8"?>
<accounts
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <account>
    <acctId>838</acctId>
    <totalDue/>
    <creationDate/>
    <accountState/>
    <bill
makeBilURI=" "/>
    <pay
makeRctURI=" "/>
    <paymentAmt/>
    <lastPaymentDate/>
    <minYearCreate/>
    <minMonthCreate/>
    <minDayCreate/>
    <maxYearPaid/>
    <maxMonthPaid/>
    <submitCollection
makeDelURI=" "/>
    <submitCredit
makeCreURI=" "/>
    <maxDayPaid/>
    <order>
      <invoiceId>316</invoiceId>
      <acctId>838</acctId>
      <sku>20950</sku>
      <quantity>69</quantity>
      <unitPrice>891.61</unitPrice>
      <discount>0.1421</discount>
      <orderDate>2001-06-26</orderDate>
      <amount/>
      <yearCreate/>
      <monthCreate/>
      <dayCreate/>
    </order>
    <order>
      ...
    </order>
    <payment>
      <acctId>838</acctId>
      <amount>1310000</amount>
      <paymentDate>2005-06-12</paymentDate>
      <yearPaid/>
      <monthPaid/>
      <dayPaid/>
    </payment>
    <payment>
      ...
    </payment>
  </account>
```

```

    <account>
    ...
  </account>
</accounts>

```

Here you have a list of accounts with each account having account specific information as well as orders and payments associated with that account. Next, you'll move onto the input XML instance data.

## Processing input

The input XML instance data for this form is more involved than previous forms because there are more local variables that you'll use in displaying reports (see Listing 15).

### Listing 15. The input XML document

```

<xforms:instance
id="atui">
  <formui
xmlns=" ">
  <MeasureLabel/>
  <CreditLabel/>
  <PaidLabel/>
  <CurrentLabel/>
  <DueLabel/>
  <PastdueLabel/>
  <DelinquentLabel/>
  <CreditGraph/>
  <PaidGraph/>
  <CurrentGraph/>
  <DueGraph/>
  <PastdueGraph/>
  <DelinquentGraph/>
  <totalAccounts/>
  <percentAccountsCredit/>
  <percentAccountsPaid/>
  <percentAccountsCurrent/>
  <percentAccountsDue/>
  <percentAccountsPastdue/>
  <percentAccountsDelinquent/>
  <zeroPercent>X</zeroPercent>
  <tenPercent>XXXXXXXXXXXX</tenPercent>
  <twentyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</twentyPercent>
  <thirtyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</thirtyPercent>
  <fourtyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</fourtyPercent>
  <fiftyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  </fiftyPercent>
  <sixtyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  </sixtyPercent>
  <seventyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXX</seventyPercent>
  <eightyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXX</eightyPercent>
  <ninetyPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXX</ninetyPercent>
  <hundredPercent>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXX</hundredPercent>
  <billStates>

```

```

<state
name="Credit"/>
<state
name="Paid"/>
<state
name="Current"/>
<state
name="Due"/>
<state
name="Pastdue"/>
<state
name="Delinquent"/>
</billStates>
<reports>
<report
value="BarGraphAccountStates"
name="Graph
Account States"
label="Graph of
Account States
(may not add up
to
100% due to
rounding):"/>
<report
value="AmountCredit"
name="Tally
Dollar Amount in
Customer Credits"
label="Total
Credits: $"/>
<report
value="AmountPastDue"
name="Tally
Dollar Amount
Past Due"
label="Total
Amount Past Due:
$"/>
<report
value="AmountDelinquent"
name="Tally
Dollar Amount
Delinquent"
label="Total
Amount
Delinquent: $"/>
</reports>
<reportSelector/>
<tally/>
</formui>
</xforms:instance>

```

The labels, graphs and percent elements are used in creating a textual representation of a bar chart. For this form the bar graph will be represented using a series of consecutive Xs. The children elements of reports specify the types of reports available. One is a bar graph; the other three are summations. The current type of report is stored in the `reportSelector` element, and headlines are displayed and the summation results are stored in `tally`, depending on the value in `reportSelector` (the type of report to be viewed).

Next, you'll jump into the fun part -- creating the form.



## The essentials

The essentials of the form, including where to put the instance data, are shown in Listing 16. Note that you should recognize several of the `bind` statements from the billing form in [Part 3](#) of this series. The same `bind` elements and calculations are needed in this form that were done in the billing form.

### Listing 16. The basic analysis 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
    Billing
    Analysis</title>
    <link
    rel="stylesheet"
    href="style.css"
    type="text/css"/>
    <style
    type="text/css">
      textarea {
        font-family:
        sans-serif;
        height: 1.2em;
        width: 90%; }
    </style>
    <xforms:model
    id="acctToolAnalyzeModel"
    >
  <xforms:instance
    id="atui">
    <formui
    xmlns="">
    <queryBilling/>
    <startDate/>
    <endDate/>
    <saveBilling/>
    <MeasureLabel/>
    <CreditLabel/>
    <PaidLabel/>
    <CurrentLabel/>
    ...
    </formui>
  </xforms:instance>
  <xforms:instance
    id="analyze"
    src="analyze_small.xml">
```

```

        <accounts
xmlns=" " />
</xforms:instance>

<xforms:submission
id="load_accts"
action="get_analyze_data.php"
method="get"
replace="instance"
instance="analyze"
ref="instance('atui') "
/>

<xforms:bind
nodeset="instance('analyze')//order/yearCreate"
calculate="substring(..orderDate,
1, 4)"/>
<xforms:bind
nodeset="instance('analyze')//minYearCreate"
calculate="min(..yearCreate)"/>
<xforms:bind
nodeset="instance('analyze')//order/monthCreate"
calculate="if(..yearCreate
=
../../minYearCreate,number(substring
(..orderDate, 6,
2)),13)"/>
<xforms:bind
nodeset="instance('analyze')//minMonthCreate"
calculate="min(..order/monthCreate)"/>
<xforms:bind
nodeset="instance('analyze')//order/dayCreate"
calculate="if(..monthCreate
=
../../minMonthCreate,number(substring
(..orderDate, 9,
2)),32)"/>
<xforms:bind
nodeset="instance('analyze')//minDayCreate"
calculate="min(..order/dayCreate)"/>
<xforms:bind
nodeset="instance('analyze')//creationDate"
calculate="concat(..minYearCreate,
'-',
if(..minMonthCreate
< 10,
concat('0',..minMonthCreate),
..minMonthCreate),
'-',
if(..minDayCreate
< 10,
concat('0',..minDayCreate),
..minDayCreate)"/>

<xforms:bind
nodeset="instance('analyze')//payment/yearPaid"
calculate="substring(..paymentDate,
1, 4)"/>
<xforms:bind
nodeset="instance('analyze')//maxYearPaid"
calculate="max(..yearPaid)"/>
<xforms:bind
nodeset="instance('analyze')//payment/monthPaid"
calculate="if(..yearPaid
=
../../maxYearPaid,number(substring
(..paymentDate,
6, 2)),0)"/>
<xforms:bind

```

```

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

<xforms:bind
nodeset="instance('analyze')//order/amount"
calculate="..//quantity
* ..//unitPrice *
(1-..//discount)"/>
<xforms:bind
nodeset="instance('analyze')//totalDue"
calculate="round(sum(..//order/amount)
- sum
(..//payment/amount)"/>
<xforms:bind
nodeset="instance('analyze')//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,
'Credit',
'Paid')"/>

<xforms:bind
nodeset="instance('analyze')//submitCollection"
relevant="..//accountState
= 'Delinquent'"/>

```

```

<xforms:bind
nodeset="instance('analyze')//submitCredit"
relevant="../accountState
= 'Credit'"/>

<xforms:bind
nodeset="instance('atui')//tally"
calculate=
"if(../reportSelector='BarGraphAccountStates',
'',
if(../reportSelector='AmountCredit',
sum(instance('analyze')//totalDue[../accountState='Credit'])
*number('-1'),
if(../reportSelector='AmountDelinquent',
sum(instance('analyze')//totalDue[../accountState='Delinquent']),
if(../reportSelector='AmountPastDue',
sum(instance('analyze')//totalDue[../accountState='Pastdue']),
'')))/>

```

The `analyze_small.xml` file contains some test data that you can use to debug and test your form with. The rest of the `bind`, `calculate` and `relevant` elements should be familiar from the billing form in [Part 3](#) of this series, except the last one, `tally`. The contents of this element change depending on the value of `reportSelector`, a `select1` element that stores what type of report is desired. Note that it's only activated for the three summation reports. If the total customer credit is chosen, `AmountCredit`, then the total due for each account with a negative balance (credit) will be summed and displayed in the `tally` element. The bar graph data is used for the first report type, which you'll see in the next section.

## The form body

The form body contains elements for displaying form data from the reports (see [Listing 17](#)).

### Listing 17. The form body

```

</xforms:model>
</head>

<body>
  Note the
  behind the scenes
  PHP in this XForm
  uses the SQL data
  and tables for
  the
  Billing XForm in
  Part 3 of this
  series.<br/>
  <xforms:submit
  submission="load_accts"
  ref="instance('atui')/queryBilling">
  <xforms:label>Refresh/load
  Billing
  Data</xforms:label>
  </xforms:submit>

```

```

    <hr />
    <xforms:select1
    ref="instance('atui')/reportSelector"
    incremental="true()">
    <xforms:label>Choose
    report type:
    </xforms:label>
    <xforms:itemset
    nodeset="instance('atui')/reports/report">
    <xforms:label
    ref="@name" />
    <xforms:value
    ref="@value" />
    </xforms:itemset>
    </xforms:select1><br/><br/><br/>

    <xforms:output
    value="instance('atui')//report[@value=instance('atui')/reportSelector]
    /@label" /><xforms:output
    ref="instance('atui')//tally" /><br/>
    <pre>
    <xforms:output
    value="instance('atui')//MeasureLabel" />
    <xforms:output
    value="instance('atui')//CreditLabel" />
    <xforms:output
    value="instance('atui')//CreditGraph" />

    <xforms:output
    value="instance('atui')//PaidLabel" />
    <xforms:output
    value="instance('atui')//PaidGraph" />

    <xforms:output
    value="instance('atui')//CurrentLabel" />
    <xforms:output
    value="instance('atui')//CurrentGraph" />

    <xforms:output
    value="instance('atui')//DueLabel" />
    <xforms:output
    value="instance('atui')//DueGraph" />

    <xforms:output
    value="instance('atui')//PastdueLabel" />
    <xforms:output
    value="instance('atui')//PastdueGraph" />

    <xforms:output
    value="instance('atui')//DelinquentLabel" />
    <xforms:output
    value="instance('atui')//DelinquentGraph" />

    </pre>
    </body>
</html>

```

Pretty simple! Most of it won't display initially, and the bar graph data will only display if that report is chosen in the above `select1` element. The bar graph is displayed within `<pre>` tags so that the bar charts line up with the measure meter (0% through 100% -- you'll see what is meant by taking another look at [Figure 2](#)). The graph labels and graphs are then displayed within the `<pre>` tags, and in the next couple sections you'll take a look at how to set them.

## Setting the bar graph labels

As shown in Listing 18, `CreditLabel`, `PaidLabel`, etc., are `calculate` attributes of the `bind` elements for the bar graph labels instance data, and they need to set the appropriate fields only when the bar graph report is selected (see Listing 18).

### Listing 18. Setting bar graph labels

```
<xforms:bind
nodeset="instance('atui')//MeasureLabel"
calculate="if(..reportSelector='BarGraphAccountStates',concat(
'
0%
10%      20%
30%      40%
50%',
'
60%
70%      80%
90%
100%'),'')"/>
<xforms:bind
nodeset="instance('atui')//CreditLabel"
calculate="if(..reportSelector='BarGraphAccountStates','Credit','')"/>
<xforms:bind
nodeset="instance('atui')//PaidLabel"
calculate="if(..reportSelector='BarGraphAccountStates','Paid','')"/>
<xforms:bind
nodeset="instance('atui')//CurrentLabel"
calculate="if(..reportSelector='BarGraphAccountStates','Current',
'')"/>
<xforms:bind
nodeset="instance('atui')//DueLabel"
calculate="if(..reportSelector='BarGraphAccountStates','Due','')"/>
<xforms:bind
nodeset="instance('atui')//PastdueLabel"
calculate="if(..reportSelector='BarGraphAccountStates',
'Past Due','')"/>
<xforms:bind
nodeset="instance('atui')//DelinquentLabel"
calculate="if(..reportSelector='BarGraphAccountStates','Delinquent',
'')"/>
```

That should do it. Next you'll calculate totals and percentages for the bar graph.

## Calculating totals and percentages

The last thing that needs to be done is summing the number of accounts (the number of `account` elements in the instance data, as shown in [Listing 14](#)), and then setting each graph's percentage depending on its piece of the overall pie (see [Listing 19](#)).

### Listing 19. Setting the bar graph data

```
<xforms:bind
```

```

nodeset="instance('atui')//totalAccounts"
calculate="count(instance('analyze')//account)"/>
<xforms:bind
nodeset="instance('atui')//percentAccountsCredit"
calculate="count(instance('analyze')//account[accountState='Credit'])
    div
    ../totalAccounts
    * 100"/>
<xforms:bind
nodeset="instance('atui')//percentAccountsPaid"
calculate="count(instance('analyze')//account[accountState='Paid'])
    div
    ../totalAccounts
    * 100"/>
<xforms:bind
nodeset="instance('atui')//percentAccountsCurrent"
calculate="count(instance('analyze')//account[accountState='Current'])
    div
    ../totalAccounts
    * 100"/>
<xforms:bind
nodeset="instance('atui')//percentAccountsDue"
calculate="count(instance('analyze')//account[accountState='Due'])
    div
    ../totalAccounts
    * 100"/>
<xforms:bind
nodeset="instance('atui')//percentAccountsPastdue"
calculate="count(instance('analyze')//account[accountState='Pastdue'])
    div
    ../totalAccounts
    * 100"/>
<xforms:bind
nodeset="instance('atui')//percentAccountsDelinquent"
calculate="
count(instance('analyze')//account[accountState='Delinquent'])
    div
    ../totalAccounts
    * 100"/>

```

Firstly, you sum the total number of accounts in the totalAccounts field. Then for each of the six account status types, you sum them all up, divide by the total number of accounts and then multiply by 100 to get a percentage. The next section takes these percentages and assigns them to graphs.

## Calculating the graphs

Now that you have percentages, you can set the graph display values. You'll have 11 buckets that each graph can fall under: zero, ten, twenty, thirty, forty, fifty, sixty, seventy, eighty, ninety, and one hundred percent, 11 available values in all. Note that this will cause rounding errors, as the data most likely won't add up to 100, but will likely exceed 100 most of the time on larger datasets. See how to set these buckets in Listing 20.

### Listing 20. Setting the graph display data

```
<xforms:bind
```

```

nodeset="instance('atui')//CreditGraph"
calculate="if(../reportSelector!='BarGraphAccountStates','',
if(../percentAccountsCredit
<= 0,
../zeroPercent,
if(../percentAccountsCredit
<= 10,
../tenPercent,
if(../percentAccountsCredit
<= 20,
../twentyPercent,
if(../percentAccountsCredit
<= 30,
../thirtyPercent,
if(../percentAccountsCredit
<= 40,
../fortyPercent,
if(../percentAccountsCredit
<= 50,
../fiftyPercent,
if(../percentAccountsCredit
<= 60,
../sixtyPercent,
if(../percentAccountsCredit
<= 70,
../seventyPercent,
if(../percentAccountsCredit
<= 80,
../eightyPercent,
if(../percentAccountsCredit
<= 90,
../ninetyPercent,
../hundredPercent)))))))))"/>

<xforms:bind
nodeset="instance('atui')//PaidGraph"
calculate="if(../reportSelector!='BarGraphAccountStates','',
if(../percentAccountsPaid
<= 0,
../zeroPercent,
if(../percentAccountsPaid
<= 10,
../tenPercent,
if(../percentAccountsPaid
<= 20,
../twentyPercent,
if(../percentAccountsPaid
<= 30,
../thirtyPercent,
if(../percentAccountsPaid
<= 40,
../fortyPercent,
if(../percentAccountsPaid
<= 50,
../fiftyPercent,
if(../percentAccountsPaid
<= 60,
../sixtyPercent,
if(../percentAccountsPaid
<= 70,
../seventyPercent,
if(../percentAccountsPaid
<= 80,
../eightyPercent,
if(../percentAccountsPaid
<= 90,
../ninetyPercent,
../hundredPercent)))))))))"/>

```

```

<xforms:bind
nodeset="instance('atui')//CurrentGraph"
calculate="if(..reportSelector!='BarGraphAccountStates','',
if(..percentAccountsCurrent
<= 0,
../zeroPercent,
if(..percentAccountsCurrent
<= 10,
../tenPercent,
if(..percentAccountsCurrent
<= 20,
../twentyPercent,
if(..percentAccountsCurrent
<= 30,
../thirtyPercent,
if(..percentAccountsCurrent
<= 40,
../fortyPercent,
if(..percentAccountsCurrent
<= 50,
../fiftyPercent,
if(..percentAccountsCurrent
<= 60,
../sixtyPercent,
if(..percentAccountsCurrent
<= 70,
../seventyPercent,
if(..percentAccountsCurrent
<= 80,
../eightyPercent,
if(..percentAccountsCurrent
<= 90,
../ninetyPercent,
../hundredPercent)))))))))"/>

<xforms:bind
nodeset="instance('atui')//DueGraph"
calculate="if(..reportSelector!='BarGraphAccountStates','',
if(..percentAccountsDue
<= 0,
../zeroPercent,
if(..percentAccountsDue
<= 10,
../tenPercent,
if(..percentAccountsDue
<= 20,
../twentyPercent,
if(..percentAccountsDue
<= 30,
../thirtyPercent,
if(..percentAccountsDue
<= 40,
../fortyPercent,
if(..percentAccountsDue
<= 50,
../fiftyPercent,
if(..percentAccountsDue
<= 60,
../sixtyPercent,
if(..percentAccountsDue
<= 70,
../seventyPercent,
if(..percentAccountsDue
<= 80,
../eightyPercent,
if(..percentAccountsDue
<= 90,
../ninetyPercent,
../hundredPercent)))))))))"/>

```

```

<xforms:bind
nodeset="instance('atui')//PastdueGraph"
calculate="if(..reportSelector!='BarGraphAccountStates','',
if(..percentAccountsPastdue
<= 0,
../zeroPercent,
if(..percentAccountsPastdue
<= 10,
../tenPercent,
if(..percentAccountsPastdue
<= 20,
../twentyPercent,
if(..percentAccountsPastdue
<= 30,
../thirtyPercent,
if(..percentAccountsPastdue
<= 40,
../fortyPercent,
if(..percentAccountsPastdue
<= 50,
../fiftyPercent,
if(..percentAccountsPastdue
<= 60,
../sixtyPercent,
if(..percentAccountsPastdue
<= 70,
../seventyPercent,
if(..percentAccountsPastdue
<= 80,
../eightyPercent,
if(..percentAccountsPastdue
<= 90,
../ninetyPercent,
../hundredPercent)))))))))"/>

```

```

<xforms:bind
nodeset="instance('atui')//DelinquentGraph"
calculate="if(..reportSelector!='BarGraphAccountStates','',
if(..percentAccountsDelinquent
<= 0,
../zeroPercent,
if(..percentAccountsDelinquent
<= 10,
../tenPercent,
if(..percentAccountsDelinquent
<= 20,
../twentyPercent,
if(..percentAccountsDelinquent
<= 30,
../thirtyPercent,
if(..percentAccountsDelinquent
<= 40,
../fortyPercent,
if(..percentAccountsDelinquent
<= 50,
../fiftyPercent,
if(..percentAccountsDelinquent
<= 60,
../sixtyPercent,
if(..percentAccountsDelinquent
<= 70,
../seventyPercent,
if(..percentAccountsDelinquent
<= 80,
../eightyPercent,
if(..percentAccountsDelinquent
<= 90,
../ninetyPercent,

```

```
../hundredPercent)))))))/>  
</xforms:model>  
</head>
```

This completes the form. Now you'll get a nice bar graph showing the percentages of customers with credit, fully paid, current, due, past due, and delinquent.

---

## Section 6. Summary

Congratulations: You completed Part 5 of the series. You now have a fully functional payables and reports form. You can track your company's liabilities. For the reports form, there are several possibilities for extending the functionality by adding other reports, or perhaps using SVG for graphics.

Come back for the final installment in Part 6 where you'll see how to integrate all of the forms and login functionality.

## Downloads

Description	Name	Size	Download method
Part 5 sample code	accttool_part5_source.zip	14KB	<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).
- To learn more about integrating XForms and graphics using SVG see the article [Creating an XForms-based logo generator](#) (Nicholas Chase, developerWorks, February 2007).
- 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

Tyler Anderson

Tyler Anderson graduated with a degree in Computer Science from Brigham Young University in 2004 and is currently in his last semester

as a Master of Science student in Computer Engineering. In the past, he worked as a database programmer for DPMG.COM, and he is currently an engineer for Stexar Corp., based in Beaverton, Oregon.

---

### 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.