

Use XForms to create an accounting tool, Part 2: Logging in and accounts

Leverage the power of XForms for user authentication

Skill Level: Intermediate

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

27 Mar 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 2 of this six-part series demonstrates how to leverage the power of XForms in conjunction with PHP and MySQL to create the basic "login," "registration," and "account management" functionality seen in many modern Web applications.

Section 1. Before you start

This tutorial is for Web application developers investigating the XForms Web form processing technology. This tutorial is the first tutorial and second installment of a [six-part series](#). The login form and account registration forms are developed in this tutorial. First, the tutorial will address some of the basics of XForms development including text entry, password obscuring text entry, and the data structures that are required. Then, the tutorial will cover the processing of the XForms data in PHP and the supporting MySQL database structures. The last part of the login is addressing what happens when a login is successful or unsuccessful. A requirement to being able to log in is having an account. The creation of accounts is made possible

through a registration process, which is covered after the login.

About this tutorial

The login form is nearly omnipresent throughout the Internet, supporting everything from e-mail access to shopping carts and even online banking. Logging in provides users data customized to their specific account and provides the system a way to control usage. Typical approaches to logging in require the user to enter a username and password that are then authenticated against a database set up prior to the login attempt. This tutorial does not purport to be an example of security in user authentication; instead, it will demonstrate the usage of XForms in the well-known algorithm for logging in and account creation.

Logging in requires the following basic process, and they will be covered in this tutorial:

1. Enter a username and password into text entry boxes.
2. Send username and password to authentication service.
3. Use the given username and account to determine if there is an account with the specified username and password given.
4. Store the result (true or false) such that all subsequent activities performed by that user are automatically associated with that authentication.
5. Allow the user access if the login succeeded or deny the user access and provide a message to indicate that the username and password could not be found.

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 knowledge is expected of the reader. There are some very good articles and introductory series concerning XForms available on developerWorks (see [Resources](#)). XForms is built on XML, and, hence, a basic understanding of XML is also assumed.

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

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

Section 2. Creating the login form

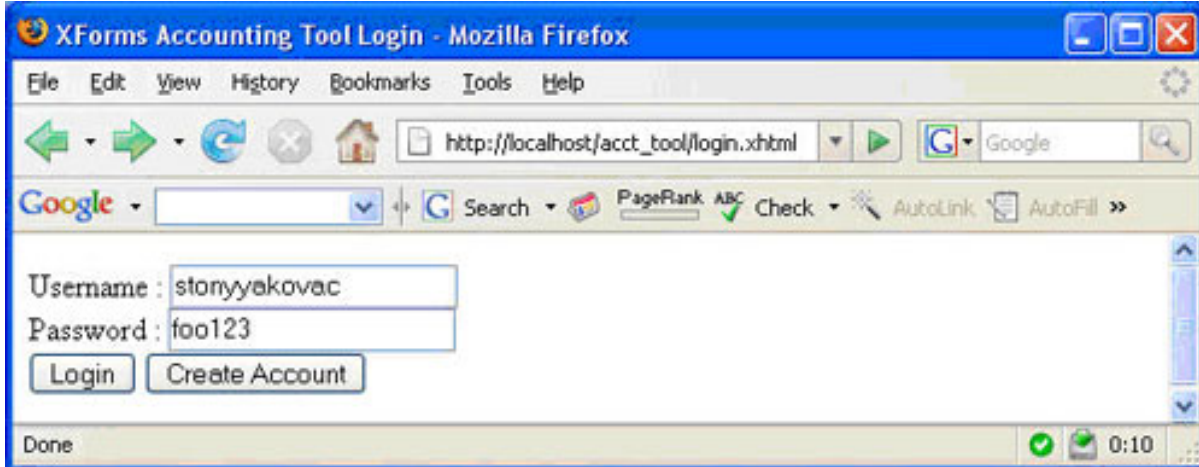
The first step in securing an application is to control access, typically through a login form. How do you keep the login form secure?

Login credential entry

Two types of login credentials exist, the username and the password. As a potential security gateway, the XForms form must offer security against a specific kind of attack. Though this tutorial does not offer guidance on strong or weak security, at

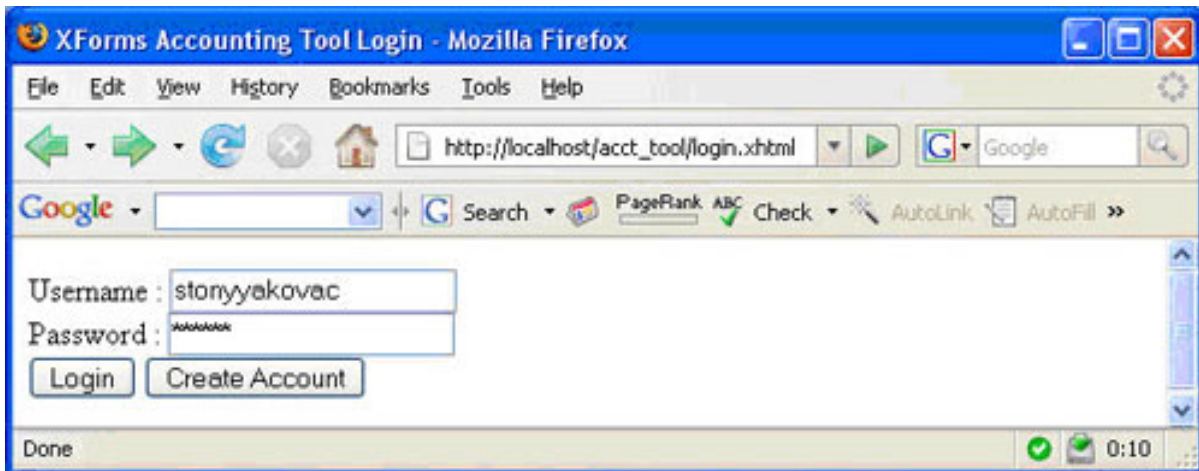
this point in time, a feature of XForms provides protection against a specific form of security attack, the Peeping-Tom attack. The Peeping-Tom attack is in-person observer who watches over the shoulder of the user and can easily gain access to the user's account with a form such as shown in Figure 1.

Figure 1. An unsecure login form



Early HTML forms offer a solution and the same basic solution exists in XForms. The preferred display to prevent the Peeping-Tom attack obscures the password, yielding a display as shown in Figure 2.

Figure 2. A Peeping-Tom secure login form



The same format and the same information exist in both forms. The data behind the forms does not differ in any way. The asterisks do not offer any protection from any one of a number of other sorts of security attacks. Some examples of such attacks are:

- The Trojan Horse attack -- An attacker creates a form that looks and acts much like the login form and saves usernames and passwords for his own use at a later date.

- The Network Intercept aAttack -- An attacker "hacks" the network and captures the data as it travels from the user's computer to the server.
- Decryption attacks -- An attacker gains access to the password text and "cracks" the encryption, reversing the algorithm and gaining the username and password pair.

To prevent such attacks requires security outside the scope of XForms. Such types of security come from encrypted protocols, firewalls, isolated computer systems, and many other methods that prevent attackers from accessing the password and username data. This example offers no significant challenge to an attacker and represents a very weak security mechanism because the password and username travel the network as plain text, easily read from the network, provided the attacker has access. The password exists in the database as plain text, which also represents extremely weak security. Though these things should be noted, the method of thwarting the Peeping-Tom should be noted as a useful feature.

Secret and input -- To obscure or not

[Listing 1](#) contains the XForms display code for displaying and obtaining the username and password text as shown in [Figure 2](#). An "input" display element serves the purpose of the username entry box. This form of display control generally creates some sort of single line entry text box. XForms technology offers independence of the form display from the architecture of the form as one of its features. Future implementations of XForms may implement the "input" display control as a verbal queue where the "label" is read to the user and the user's response is converted to text before submission. The password control behaves exactly like the "input" control except the characters are echoed to the screen as asterisks instead of the actual typed characters. For each of the two form controls, a "label" specifies an indicator of the nature of the data to be entered in the control. Most XForms controls allow the programmer to specify a label and use it in some unspecified, but reasonable, way.

Listing 1. The username and password display controls

```
<xforms:input
ref="instance('acctToolLoginInst')/username"
>
<xforms:label>Username
: </xforms:label>
</xforms:input><br/>
<xforms:secret
ref="instance('acctToolLoginInst')/password"
>
<xforms:label>Password
: </xforms:label>
</xforms:secret><br/>
```

Linking the form control to storage

Listing 1 contains very little information other than the fact that an "input" and a "secret" should be rendered. The only other interesting piece of information is the `ref`. Specifiers contained within the "<" and ">" tokens are properties or attributes. XForms allows the insertion of nearly anything as an attribute in this way; however, only specific attributes have meaning. Every form control that serves a purpose uses either the `ref` or the `bind` attribute. Either can perform identically. The difference between the two is that `bind` refers to a statement that will be discussed in more detail later in this tutorial while `ref` refers to an XML data node.

XPath -- Finding the data storage

XForms is built on XML; it is the data structure behind the presentation. Through this structure, XForms can change its presentation or save the data entered in the form. The string specified by "ref" is known as XPath. XPath can be as simple as a hierarchical reference to something such as the string "/top/down1/hereitis," or as complicated as a programmer can dream up without using a second line of code. At this point, the one-liner obfuscated code champions of the world rejoice and cheer. Now there is a reason to see what can be done on one line. A small subset of XPath functions is available to XForms. The "instance" function serves a critical role in XForms.

Instances -- XForms data containers

XForms requires an XML data structure to live behind each form control. That XML data structure, in turn, lives inside an instance. That instance lives inside an XForms model. A single XHTML page may contain an unspecified number of XForms models and each XForms model may contain an unspecified number of instances. The `instance` XPath function provides a mechanism for finding a specific instance. From that point, the rest of the path utilizes the instance as the root and locates a node relative to that instance. Listing 2 shows the instance definition behind the login form.

Listing 2. The instance XML data structure

```
<xforms:instance
id="acctToolLoginInst"
>
<loginForm
xmlns=" " >
<username>stonyyakovac</username>
<password>fool23</password>
<message/>
<menu/>
```

```
<submitLogin/>
</loginForm>
</xforms:instance>
```

Several notable features appear in Listing 2. The keyword "instance" encompasses an instance. The keyword "id" specified as an attribute of "instance" defines the name of that particular instance. The "id" specified in Listing 2 identically matches the contents of the `instance()` XPath function from Listing 1. This mechanism links the form controls with this specific instance data. The next notable feature of Listing 2 is the `xmlns` attribute to `loginForm`. `loginForm` is not a keyword; that word can be defined by the programmer as any legal text. The `loginForm` serves as a node reference point for the rest of the data. Each XForms instance may have exactly 1 child node. In this case, the child node is `loginForm`. Notice that `loginForm` does NOT appear in the XPath reference to this node. The `xmlns` attribute specifies the namespace of the instance. In this case, the form is built with no namespace. If this namespace attribute were missing, all the form controls would disappear.

Another thing to note about the instance data structure is the presence of an initializer for "username" and "password." The real implementation of a login form would not contain these initializers, but, in this case, they serve as a shortcut for debugging purposes with the effect of pre-loading the form controls with the username "stonyyakovac" and the password "foo123." The "menu," "message," and "submitLogin" nodes all demonstrate uninitialized instance data.

How buttons happen

As previously stated, the XForms implementation determines the presentation of the form controls; however, this presentation includes buttons. The buttons in question, shown in Figure 1 and Figure 2, contain the text "Login" and "Create Account". Generally, users understand that a button implies an action. These two buttons occur because of different specifying text in the XForms definition, yet they are rendered the same. Listing 3 shows the code responsible for the two buttons.

Listing 3. Submit and trigger -- The buttons

```
<xforms:submit
  submission="submit_login"
  ref="instance('acctToolLoginInst')/submitLogin">
  <xforms:label>Login</xforms:label>
</xforms:submit>
<xforms:trigger
  ref="instance('acctToolLoginInst')/newAcct">
  <xforms:label>Create
  Account</xforms:label>
  <xforms:load
  ev:event="DOMActivate"
  resource="register.xhtml"/>
</xforms:trigger>
```

Trigger -- The create account button

The Create Account button acts completely independent of the form data in this case. The "trigger" definition in Listing 3 specifies a label and another line, the `load` action. XForms implements trigger as a sort of empty socket that the developer must plug something into. This "trigger" has a `load` action in it. `load` does what the name suggests, it loads a URL specified by the attribute `resource`. A key to making this button perform as a button should, meaning that the `load` happens when the button is clicked, is the presence of the `ev:event` attribute. The `ev` portion of that attribute preceding the ":" references a namespace that was defined earlier in the code in the `html` element. The listing for these definitions is shown in Listing 4. This listing also explains why every XForms element has "xforms" prepended to it. The word inside the quotes, `DOMActivate`, instructs the "load" to happen only when the enclosing element is activated. In this case, the enclosing element is the trigger, so the load action happens when the button is clicked.

Listing 4. Namespace definition

```
<!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:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xlink="http://www.w3.org/1999/xlink"
>
```

Submit -- The login button

Generally Web forms submit data to some destination. Sometimes custom CGI scripts process the information such as checking out of a shopping card; sometimes other things happen. This form has a single submission mechanism, though any number of submission specifications may be specified. The "Login" button links to the submission mechanism through a special XForms element that must be declared inside the XForms model declaration. The "submit_login" ID links the button to the action defined in Listing 5.

Listing 5. Submission definition

```
<xforms:submission
```

```
id="submit_login"
action="login.php"
method="post"
instance="acctToolLoginInst"
replace="instance"
ref="instance('acctToolLoginInst')"
/>
```

The XForms submission declared in Listing 5 demonstrates the use of some key attributes to the success of the login form. First, though the mundane attributes, "id," "action," and "method" define that when the submission referencing "submit_login" (the login button mentioned a moment ago) is clicked, the URL "login.php" will be activated as a script using the `post` format for data exchange. There is a required reference to an XML node in the XForms instance defined by `ref`, which will become more important in a moment. The last two attributes to discuss are `instance` and `replace`. The two attributes, working in conjunction, instruct the XForms processor to use the data returned by the PHP script to replace the data labeled with the XForms ID "acctToolLoginInst."

Section 3. PHP credential authentication

The submit mechanism invokes a PHP script to use the XForms form data and return data. The PHP script performs the following basic steps:

- Find the user data.
- Look in the database to see if the data is valid.
- Return a decision and the data associated with that decision.

This section will look at how to accept XForms data in PHP, how to parse and decipher it, how to access the database, and how to react to both correct and incorrect logins.

Handling XML XForms data in PHP

XForms uses XML as its preferred data exchange format. Submission data arrives in the form of an XML document. The first listing of PHP (Listing 6) demonstrates the retrieval of the XML data, parsing of the XML data, and the access methods to that data.

Listing 6. Handling XML form data with PHP

```
<?php
    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);
    $pass =
    $doc->getElementsByTagName("password")->item(0)->nodeValue;
    $uname =
    $doc->getElementsByTagName("username")->item(0)->nodeValue;
```

By using the `DomDocument` class that comes built in with PHP5, XML becomes easy to interface. The line referencing "file_get_contents" is a safety measure in case the data only exists on the "standard in" of the PHP instead of automatically being placed into the `HTTP_RAW_POST_DATA` variable. The `DomDocument` method, `loadXML`, parses the XML data and saves it in a data structure that can be queried as shown with the `getElementsByTagName` method.

Initializing the MySQL database

The next step is to "Look in the database to see if the data is valid." Before that can be done, the database must be created. The code for this tutorial requires a MySQL server running on "localhost," the "root" account with no password, a database named "acct1," and a table named "contact." Using a MySQL database with the root account enabled and having no password represents a security hole and is discouraged, but for development purposes, it makes debugging easier. Installing and running a MySQL server on "localhost," the machine the browser is executing on, is well documented on the Web. By default, the initial configuration generally includes a root account with no password. To create the acct1 database and the required contact table, log into the MySQL server and execute the following commands (see Listing 7).

Listing 7. Initializing the MySQL database

```
CREATE DATABASE
'acct1' ;
CREATE TABLE
'contact' (
'contactId' INT(
11 ) NOT NULL ,
'username'
VARCHAR( 20 ) NOT
NULL ,
'password'
VARCHAR( 20 ) NOT
NULL ,
'prefix' VARCHAR(
```

```

10 ) NOT NULL ,
'firstName'
VARCHAR( 20 ) NOT
NULL ,
'lastName'
VARCHAR( 20 ) NOT
NULL ,
'company'
VARCHAR( 20 ) NOT
NULL ,
'addr1' VARCHAR(
40 ) NOT NULL ,
'addr2' VARCHAR(
40 ) NOT NULL ,
'city' VARCHAR(
20 ) NOT NULL ,
'zip' VARCHAR( 10
) NOT NULL ,
'state' VARCHAR(
2 ) NOT NULL ,
workNum' VARCHAR(
12 ) NOT NULL ,
'homeNum'
VARCHAR( 12 ) NOT
NULL ,
'faxNum' VARCHAR(
12 ) NOT NULL ,
'cellNum'
VARCHAR( 12 ) NOT
NULL ,
'notes' VARCHAR(
400 ) NOT NULL
) ENGINE =
innodb;
INSERT INTO
'contact' (
'username' ,
'password' ,
'firstName' ,
'lastName' ,
'addr1' ,
'addr2' , 'city'
, 'zip' , 'state'
, 'homeNum' ,
'faxNum' ,
'cellNum' ,
'contactId' ,
'company' ,
'workNum' ,
'notes' ,
'prefix' )
VALUES (
'stonyyakovac' ,
'fool23' ,
'Stony' ,
'Yakovac' , '1223
NW Rd.' , '' ,
'PDX' , '97117' ,
'OR' ,
'503-475-1411' ,
'' , '' , '12345' ,
'Yakovac
Enterprises' , '' ,
'Home: Anytime' ,
'Mr.'
);

```

Those commands will create the required database and table and insert an entry to use as a test account.

Interfacing MySQL in PHP

The PHP code for logging into the MySQL server, selecting the database, building a query, executing the query, and evaluating the results is shown in Listing 8. This process is fairly mundane and could be performed in different ways, but the point of the task is to query the database.

Listing 8. Querying the MySQL database for login credentials

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

$sqlselldb =
mysql_select_db('acct1',
$sqldb);
if
(!$sqlselldb)
die
('Can\'t select
the acct1
database
localhost
because: ' .
mysql_error());

$sqlQuery =
sprintf("SELECT 1
FROM contact
WHERE
username='%s' and
" .
"password='%s'",
mysql_real_escape_string($uname),
mysql_real_escape_string($pass));
$queryData =
mysql_query($sqlQuery,
$sqldb);
if
(!$queryData)
die ('Could
not insert token
into tokentable
on because: ' .
mysql_error());

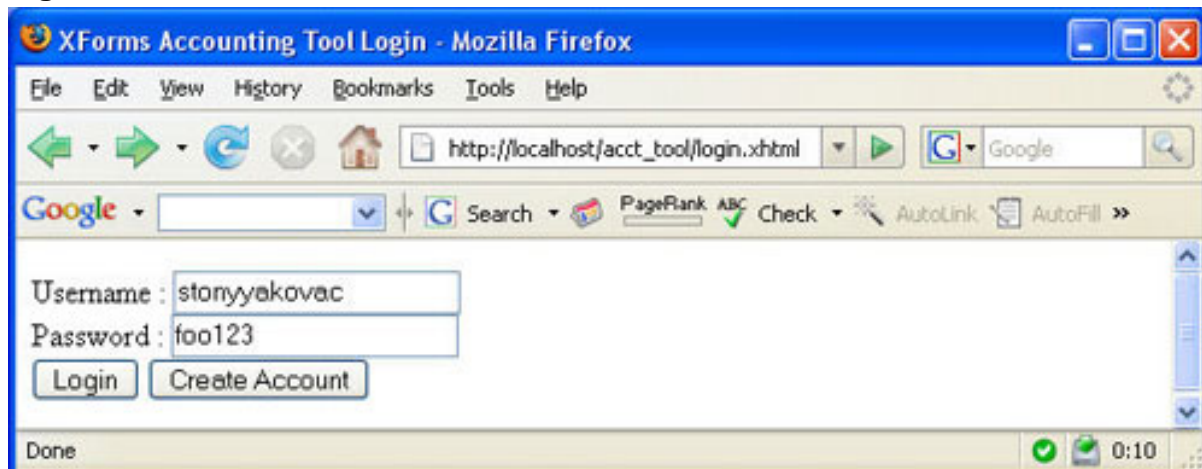
$numRows =
mysql_num_rows($queryData);
```

The best information that should be returned in the case of a failed login attempt is that the credentials were invalid. Though it might be tempting to tell the user the password was incorrect, that reaction invites a methodical brute-force password cracking attempt where a hacker first tries usernames until the PHP tells him the password is wrong, then tries passwords knowing the account is valid. Note that the data returned from the SQL query is not scalar data and a function must be used to evaluate the result.

Note: Although it's beyond the scope of this tutorial, you should check the [Resources](#) for places to get information on preventing SQL injection attacks, which commonly target applications like this.

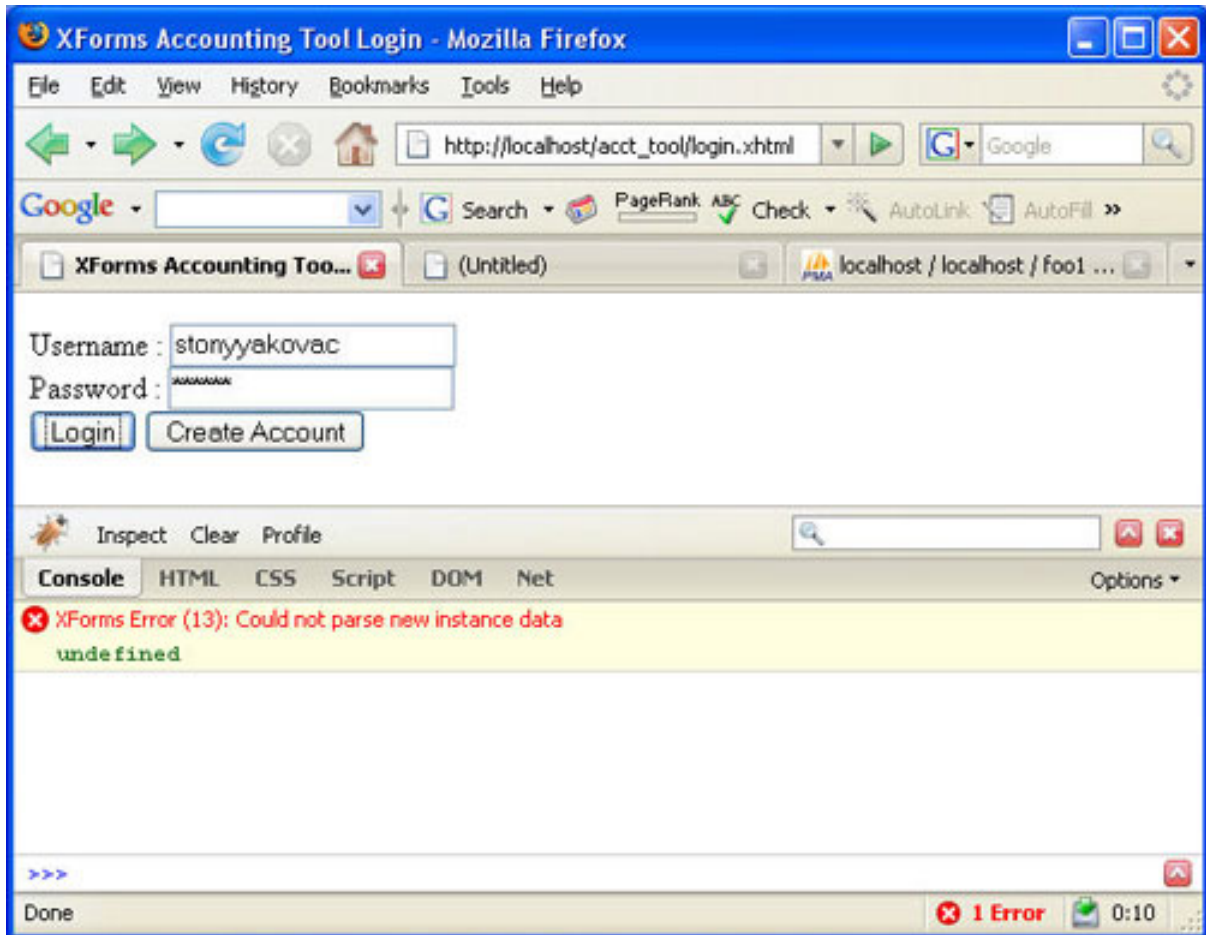
At this point, though the PHP code looks reasonable and even functions correctly, a problem exists. The "die" statements return text. Since the login.php script uses the "instance/replace" attributes on the submission, XForms assumes that all data returned from the PHP script is in XML form. Hence, if one of the "die" cases executes, the result will be that Firefox will show something like what is shown in Figure 3.

Figure 3. Firefox XForms error



The secret is in the lower right-hand corner. The red error can be unfolded to reveal the real problem, as shown in Figure 4.

Figure 4. Firefox XForms error expanded

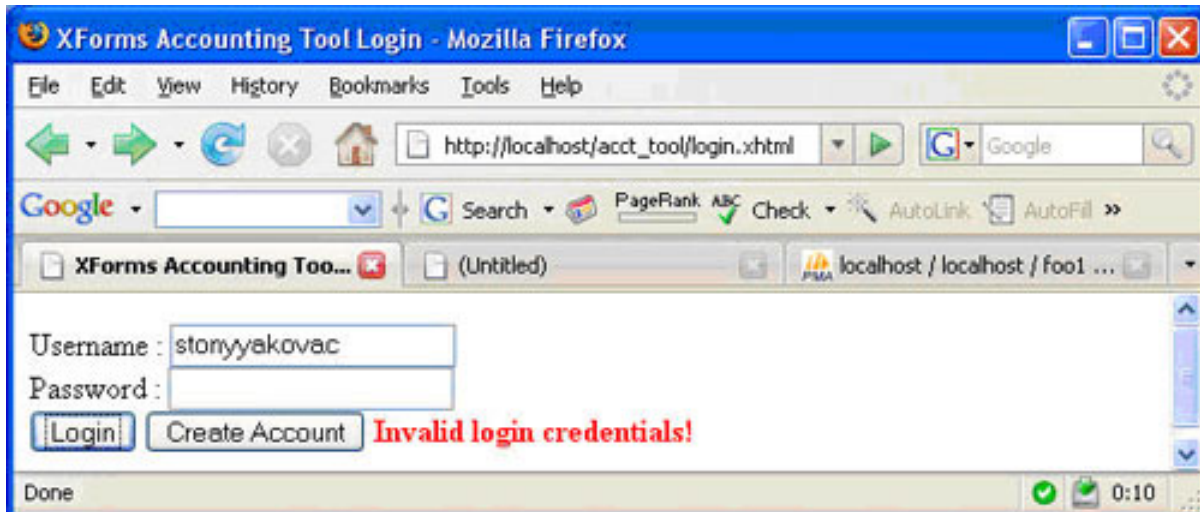


The lesson to be learned from this incredibly vague error is that only correctly formatted XML data can be consumed by the XForm. Without the "replace/instance" attributes, the error would appear in the browser window as the text in the PHP script.

Returning data for invalid logins

When a user enters the incorrect credentials, either username or password, the appropriate action is to tell the user his credentials were invalid. Without a clue that the user has typed something incorrectly, he is likely to click "Login" with frustration a few times, then give up without re-typing the password and/or username since he would assume the button doesn't work. By displaying a message such as that shown in Figure 5, the user is informed that the login data is incorrect and may fix the issue before clicking "Login" again.

Figure 5. Invalid login credentials



To return that data, a node is filled which was left blank in the original, the message node. Listings 9 and 10 show the pertinent lines of the XHTML and CSS to display the message shown in Figure 5. The "output" element of XForms displays data without the data serving as some form of input control.

Listing 9. Error message XForm code

```
<xforms:output
class="errorMessage"
ref="instance('acctToolLoginInst')/message"
/>
```

Listing 10 shows the pertinent lines of the CSS.

Listing 10. CSS to format the error message

```
*.errorMessage {
font-weight:
bold; color: red;
}
```

The next thing to notice about the form after an invalid login attempt is the missing password. The username stayed, but, to insure that the user actually re-types the password, that data is removed from the instance data. Since the submission uses "replace/instance," setting the password node to an empty string accomplishes the task. The PHP code listing is shown in Listing 11. The only other new PHP is the "saveXML" method. Activated without a parameter, the "saveXML" method returns correctly formatted XML representing its current structure, which is very convenient since XForms needs exactly that.

Listing 11. PHP invalid login clause

```

        if($numRows <=
0)
    {
mysql_close($sqldb);
$doc->getElementsByTagName("password")->item(0)->nodeValue
= '';
$doc->getElementsByTagName("message")->item(0)->nodeValue
= 'Invalid login
credentials!';
        echo
$doc->saveXML();
        exit;
    }

```

Returning data for valid logins

When the user provides a username and password that result in a query return of more than zero rows, the user meets the login criteria. Having met the criteria, the PHP needs to provide some services (see Listing 12).

- Remember the login
- Allow the user access to the tools

Listing 12. PHP valid login clause

```

        else
        {
$_SESSION['username']
= $uname;
        $loginToken
= rand();
$_SESSION['loginToken']
= $loginToken;
        $now =
localtime();
        $datetime =
sprintf("%4d-%2d-%2d
%2d:%2d:%2d", $now[5]+1900, $now[4], $now[3], $now[2], $now[1], $now[0]);
        $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);

    //$host=
    $_SERVER['HTTP_HOST'];
    //$reldir=
    rtrim(dirname($_SERVER['PHP_SELF']),
    '/\\');
    //header("Location:
    http://$host$reldir/manageAcct.xhtml");
    $notLoggedIn =
    $doc->getElementsByTagName('notLoggedIn')->item(0);
    $doc->getElementsByTagName('loginForm')->item(0)->removeChild($notLoggedIn);
    $menu =
    $doc->getElementsByTagName("menu")->item(0);
    $menu->setAttribute('title',
    "Great! You're
    logged in! Here
    are your".
    " available
    links:");
    $links =
    $doc->createElement('links');
    $menu->appendChild($links);
    $link =
    $doc->createElement('link');
    $link->setAttribute('name',
    'Account
    management');
    $link->setAttribute('value',
    "manageAcct.xhtml");
    $links->appendChild($link);
    $link =
    $doc->createElement('link');
    $link->setAttribute('name',
    'Invoices');
    $link->setAttribute('value',
    "invoice.xhtml");
    $links->appendChild($link);

    //echo
    str_replace('<',
    '<',
    str_replace('>',
    '>',
    $doc->saveXML()));
    echo
    $doc->saveXML();
    exit;
}

```

This section of PHP code first references the `$_SESSION` variable. The implementation of this feature guarantees that the data stored in the `$_SESSION` variable will always be available when the same client instance returns. This is how a login is remembered as having happened. The second part of remembering that a user has already logged in is generating a unique token and giving a copy to the user using the `$_SESSION` variable and saving a copy in the database. This token allows the PHP to insure the user correctly logged in for future transactions, as the PHP can compare the token the user reports possessing with the token in the

database for that user. The storage of a timestamp enables a timed server-based service to clean the database of old logins requiring users to re-login after some appropriate amount of time.

Before the last two portions of the PHP script are discussed, the commented lines deserve a note. Generally, most login forms, when satisfied, automatically sweep the user off to the protected pages. This is done with a redirect, as shown in the commented code. Just like the "die" statements, the redirection does not function properly, either. XForms requires that the PHP either return only XML data or only HTML data, but not whatever suits the PHP at the time. There are mechanisms in the specification of XForms that provide for the same sort of dynamic re-direction, but, as of the writing of this tutorial, the implementation does not appear to function correctly for that activity. As a substitute, a menu is created and another XForms feature creates the illusion of re-direction.

Using the `appendChild`, `setAttribute`, and `createElement` methods, XML nodes are added with enough data to supply an implementation of a button, as in the "Create Account" button covered earlier. Using the similar methods, the "notLoggedIn" node is removed.

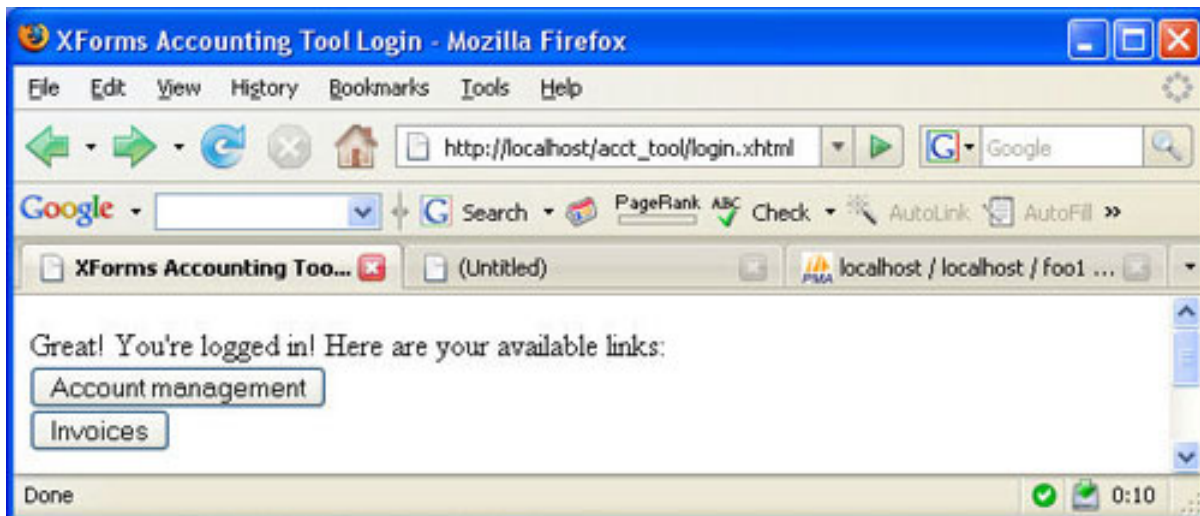
Displaying data for valid logins

The XForms code shown in Listing 13 creates the menu shown in Figure 6. The new additions to the already explained XForms mechanics are the "repeat" and, more subtle, some of the XPath expressions used in the "repeat."

Listing 13. Creating the menu for a valid login

```
<xforms:label
ref="instance('acctToolLoginInst')/menu/@title"
/>
<xforms:repeat
nodeset="instance('acctToolLoginInst')//link">
<xforms:trigger>
<xforms:label
ref="@name" />
<xforms:load
ev:event="DOMActivate"
ref="@value" />
</xforms:trigger>
</xforms:repeat>
```

Figure 6. XForms login valid menu



The XForms "repeat" structure offers leverage to create many different dynamic structures. Future tutorials in this series discuss leveraging "repeat" more, but this example is a good starting point. The "repeat" structure processes all nodes in an XML data structure referenced by the "nodeset" attribute attached to it. In this case, the nodeset is all nodes named "link." Those nodes were defined in the PHP script in [Listing 11](#).

Two new syntactical elements that make this example work are the two XPath sections in bold in [Listing 12](#). The "@" symbol tells the XPath processor to find an attribute by the name following the "@" instead of trying to find an element. That makes sense when the PHP is referenced where the data was inserted as attributes. The last is the "//", which specifies "this node and all of its descendants." Without the double slash, the links will not show.

Not displaying data before valid logins

Knowing the code for a working menu exists, the reader might have asked why the menu didn't show up in the first place. The answer to that question is trivial; the data did not exist prior to a valid login. Still, a perfectly valid question needs answering. Why did the login form controls disappear? The answer lies in one last piece of XForms code, shown in [Listing 14](#). [Listing 14](#) introduces the `bind` element of the XForms language. The `bind` statement exists in the definition of the XForms model. Its purpose is to provide a method of dynamically altering the properties of XForms controls and controlling XForms behavior. This listing uses the attribute `relevant`.

Listing 14. XForms hides the login menu

```
<xforms:bind
nodeset="instance('acctToolLoginInst')/newAcct"
relevant="instance('acctToolLoginInst')/notLoggedIn"/>
```

```
<xforms:bind
nodeset="instance('acctToolLoginInst')/submitLogin"
relevant="instance('acctToolLoginInst')/notLoggedIn"/>

<xforms:bind
nodeset="instance('acctToolLoginInst')/username"
relevant="instance('acctToolLoginInst')/notLoggedIn"/>

<xforms:bind
nodeset="instance('acctToolLoginInst')/password"
relevant="instance('acctToolLoginInst')/notLoggedIn"/>

<xforms:bind
nodeset="instance('acctToolLoginInst')/message"
relevant="instance('acctToolLoginInst')/notLoggedIn"/>
```

Each bind specifies a nodeset from the instance data and references the `notLoggedIn` node of the instance data for the relevant property. The `notLoggedIn` node is declared in the instance data; therefore, it exists when this form is first loaded. Because it exists, the node referenced by the `nodeset` attribute becomes "relevant." When a valid login occurs, the PHP script removes the `notLoggedIn` node, making all those nodes irrelevant. When a node that is connected to an XForms control is irrelevant, the XForms engine does not render that control. Using this method, the login form is hidden when a valid login is accomplished.

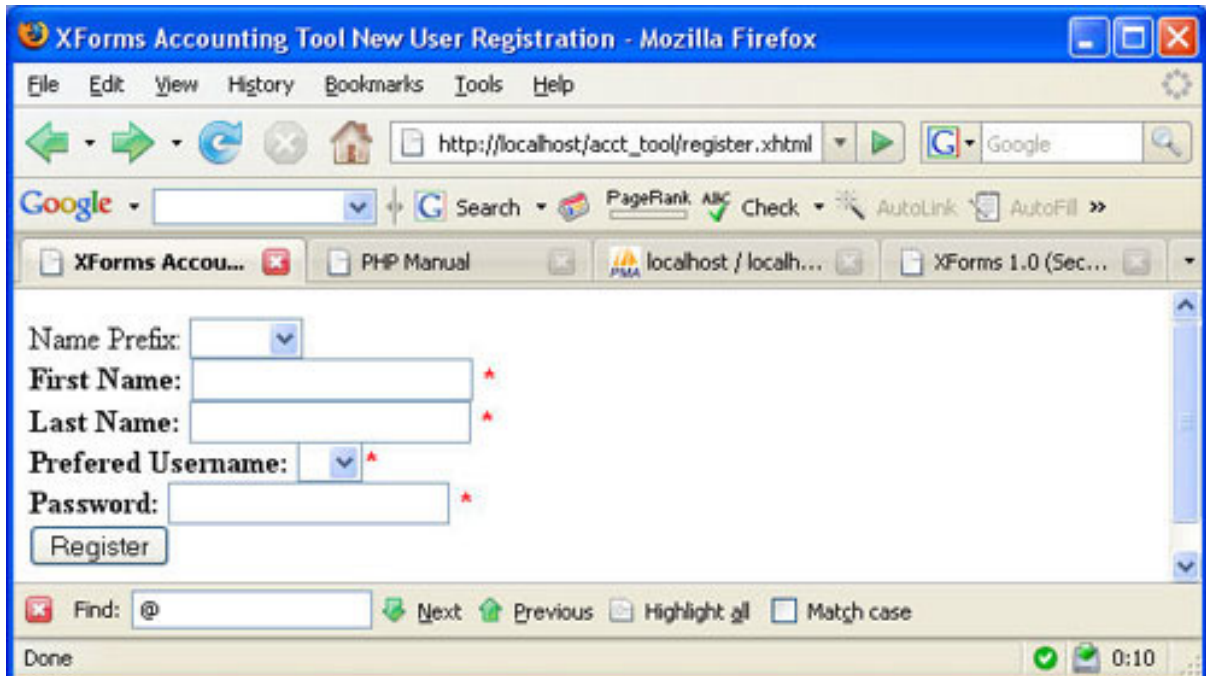
Section 4. Account registration

In this section, you will look at a new form, the user registration form. It includes a new control, the drop-down menu, and new bind controls.

Creating an account

Logging in requires an account. Creating an account requires a registration form. Figure 7 shows the registration form for this tutorial, containing very simple information at this point. Some interesting XForms features are used to advantage creating this form, however.

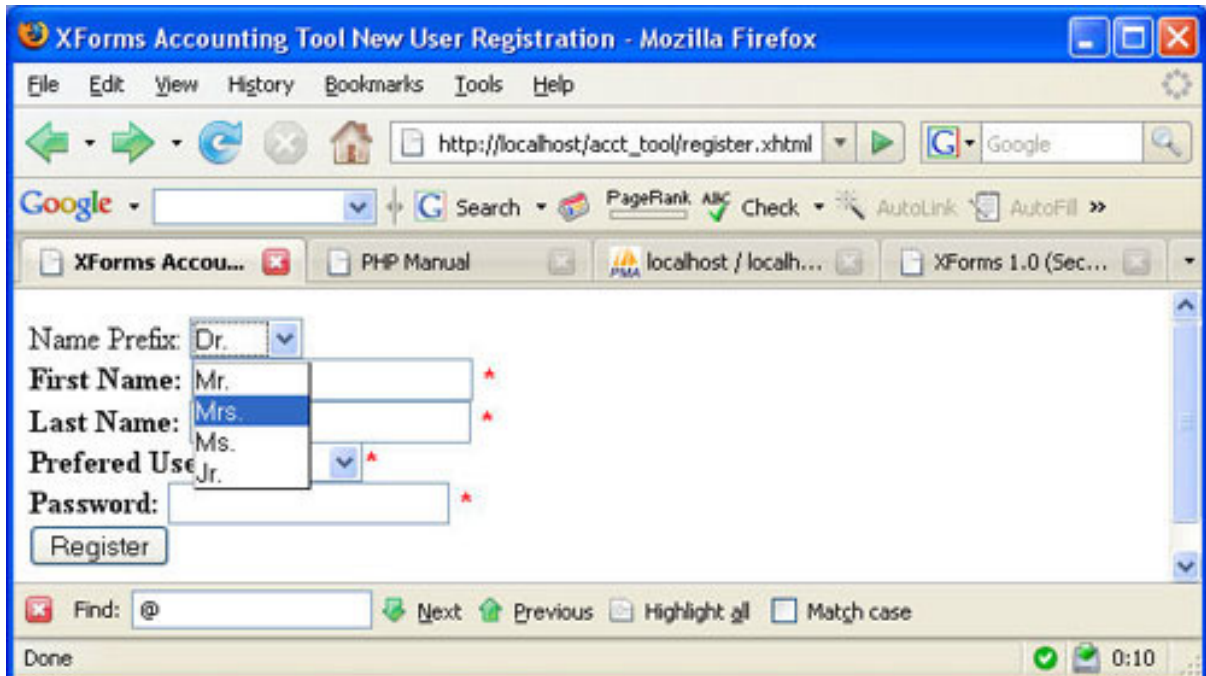
Figure 7. The basic registration form



Open selections

An open selection is used for the two drop-down selections in this form. The two boxes behave differently, however. The Name Prefix box contains a hard-coded set of options to start with. The "open" part of the selection allows the user to enter anything in the box as if it were a text-entry box. This behavior is shown in Figure 8.

Figure 8. The open selection



The XHTML to create the open selection is shown in Listing 15. A "select1" form control makes the user select exactly one item from a list. The list is constructed of any number of the "item" blocks, as seen in Listing 14. The default for a "select1" selection is "closed," which disallows user entry of a choice not in the list. The selection attribute allows the XForms form designer to create an open selection.

Listing 15. XForms Open Select1

```
<xforms:select1
ref="instance('acctToolRegInst')/prefix"
selection="open">
<xforms:label>Name
Prefix:
</xforms:label>
<xforms:item>
<xforms:label>Mr.</xforms:label>
<xforms:value>Mr.</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Mrs.</xforms:label>
<xforms:value>Mrs.</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Ms.</xforms:label>
<xforms:value>Ms.</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>Jr.</xforms:label>
<xforms:value>Jr.</xforms:value>
</xforms:item>
</xforms:select1><br/>
```

Dynamic selections

This form contains a dynamic selection example. As the user types, the username selection box expands. A new feature of XForms creates this effect, calculate. Listing 16 shows the portions of the XForms code that create this effect. The keys elements are that the items get their labels and values from the data nodes through the `ref` attribute and the data nodes get their values through the `calculate` property of the `bind` element. The three selections are three common incantations of usernames. The typical incantations in question are the concatenation of the first and last names, the first initial and last name, and the last initial and first name.

Listing 16. XForms Open Select1 And Calculate

```
<!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"
>
  <head>
    <title>XForms
Accounting Tool
New User
Registration</title>
    <link
rel="stylesheet"
href="style.css"
type="text/css"/>
    <xforms:model
id="acctToolRegModel"
>
  <xforms:instance
id="acctToolRegInst"
>
  <registerForm
xmlns="">
  <password/>
  <username/>
  <unameopt1/>
  <unameopt2/>
  <unameopt3/>
  <prefix/>
  <firstName/>
  <lastName/>
  <submitRegistration/>
</registerForm>
</xforms:instance>
  ...
  <xforms:bind
nodeset="instance('acctToolRegInst')/unameopt1"
calculate="translate(concat(concat(substring(../firstName,1,1),
../lastName), 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'abcdefghijklmnopqrstuvwxyz'))"
"/>
<xforms:bind
```

```

nodeset="instance('acctToolRegInst')/unameopt2"
calculate="translate(concat(substring(..lastName,1,1),
../firstName),'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'abcdefghijklmnopqrstuvwxyz')
"/>
<xforms:bind
nodeset="instance('acctToolRegInst')/unameopt3"
calculate="translate(concat(..firstName,
../lastName),
'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'abcdefghijklmnopqrstuvwxyz')
"/>
...
<xforms:select1
class="required"
bind="usernamebind"
>
<xforms:label>Preferred
Username:
</xforms:label>
<xforms:item>
<xforms:label
ref="instance('acctToolRegInst')/unameopt1"/>
<xforms:value
ref="instance('acctToolRegInst')/unameopt1"/>
</xforms:item>
<xforms:item>
<xforms:label
ref="instance('acctToolRegInst')/unameopt2"/>
<xforms:value
ref="instance('acctToolRegInst')/unameopt2"/>
</xforms:item>
<xforms:item>
<xforms:label
ref="instance('acctToolRegInst')/unameopt3"/>
<xforms:value
ref="instance('acctToolRegInst')/unameopt3"/>
</xforms:item>
</xforms:select1><br/>

```

Notice the introduction of more XPath functions, the `concat`, `substring`, and `translate` functions. The `translate` function converts all the uppercase characters to lowercase characters by replacing each character from the second argument with the corresponding character in the third argument. The result is shown in Figure 9. The `concat` function returns a single string result given N strings as arguments, and the `substring` function returns a specified number of characters indexed by a specified position from a source string. Notice also that the XPath location of the `firstName` and `lastName` is prefaced with `../`. This is because, when the `calculate` function is computed, it is computed relative to the node it is bound to. Looking at the instance definition, it is easy to see that the `unameopt1` and `lastName` are siblings and hence, the path is up one (`../`) and down (`lastName`).

Figure 9. The dynamic selection

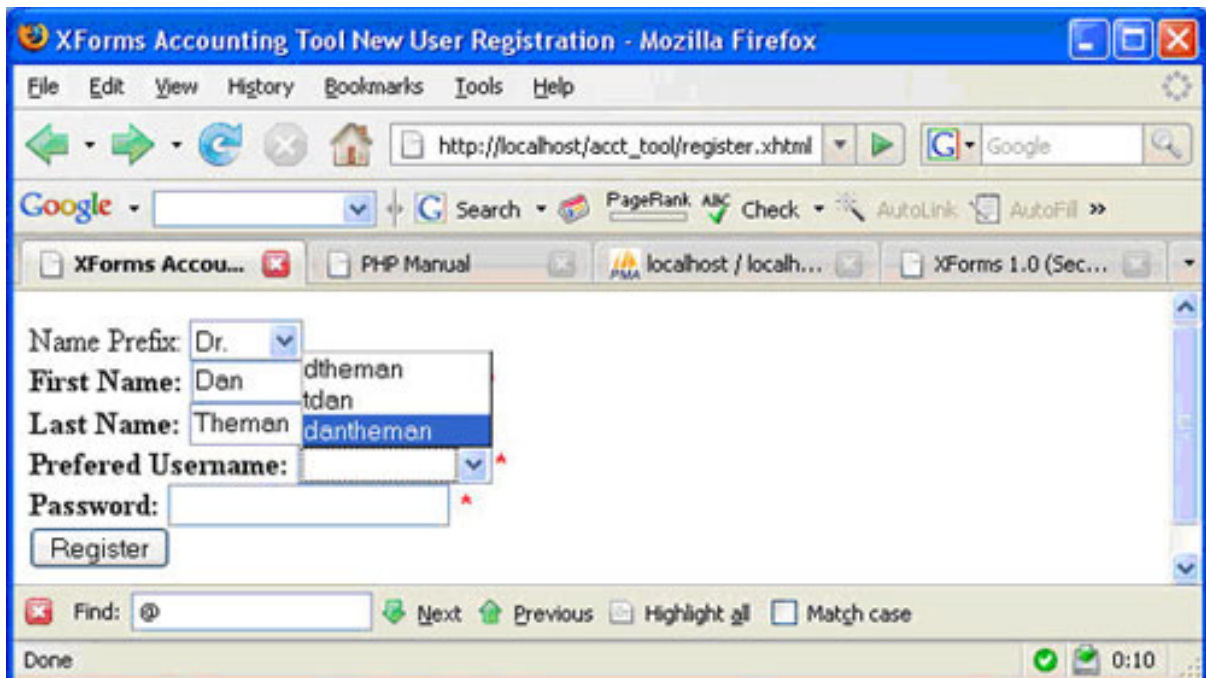


Figure 10 shows the registration form filled out. When the **Register** button is clicked, instead of the behavior seen in the login form, this form whisks the user away to different URL, the login form. The two portions of the design responsible for this behavior are the "submission" XForms definition seen in Listing 17, and the PHP code shown briefly in Listing 18.

Listing 17. XForms registration submission

```
<xforms:submission
id="submit_registration"
action="register.php"
method="post"/>
```

Listing 18 displays the registration PHP code.

Listing 18. Registration PHP

```
<?php
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);

$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 acct1
database on MySQL
server @
localhost
because: ' .
mysql_error());

    // first check
to see if that
username is
already taken
    $uname =
$doc->getElementsByTagName("username")->item(0)->nodeValue;
    $sqlQuery =
sprintf('SELECT *
FROM `contact`
WHERE
username=\'%s\'',
mysql_real_escape_string($uname));
    $queryData =
mysql_query($sqlQuery,
$sqldb);
    if
(!$queryData)
        die ('Could
not query the
contact table in
the acct db
because: ' .
mysql_error());

if(mysql_num_rows($queryData)
> 0)
{
mysql_close($sqldb);
$doc->getElementsByTagName("message")->item(0)->nodeValue
=
    'The
username ' .
$uname . ' is
already taken.
Choose another.';
    echo
$doc->saveXML();
    exit;
}
else
{
    $pass =
$doc->getElementsByTagName("password")->item(0)->nodeValue;
    $fname =
$doc->getElementsByTagName("firstName")->item(0)->nodeValue;

```

```

        $prefix =
$doc->getElementsByTagName("prefix")->item(0)->nodeValue;
        $lname =
$doc->getElementsByTagName("lastName")->item(0)->nodeValue;
        $sqlQuery =
"INSERT INTO
`contact` (
`username`,
`password`,
`firstName`, ".
        "
`lastName`,
`prefix`) VALUES
( ".
sprintf("%s",
's', 's', 's', 's',
's' );",
$username, $pass,
$fname, $lname,
$prefix);
        $queryData
=
mysql_query($sqlQuery,
$sqldb);
        if
(! $queryData)
            die
('Could not
insert token into
contact table on
MySQL server
because: '
.
mysql_error());
mysql_close($sqldb);
        $host=
$_SERVER['HTTP_HOST'];
$reldir=
rtrim(dirname($_SERVER['PHP_SELF']),
'/\');
        header("Location:
http://$host$reldir/login.xhtml");
exit;
    }
?>

```

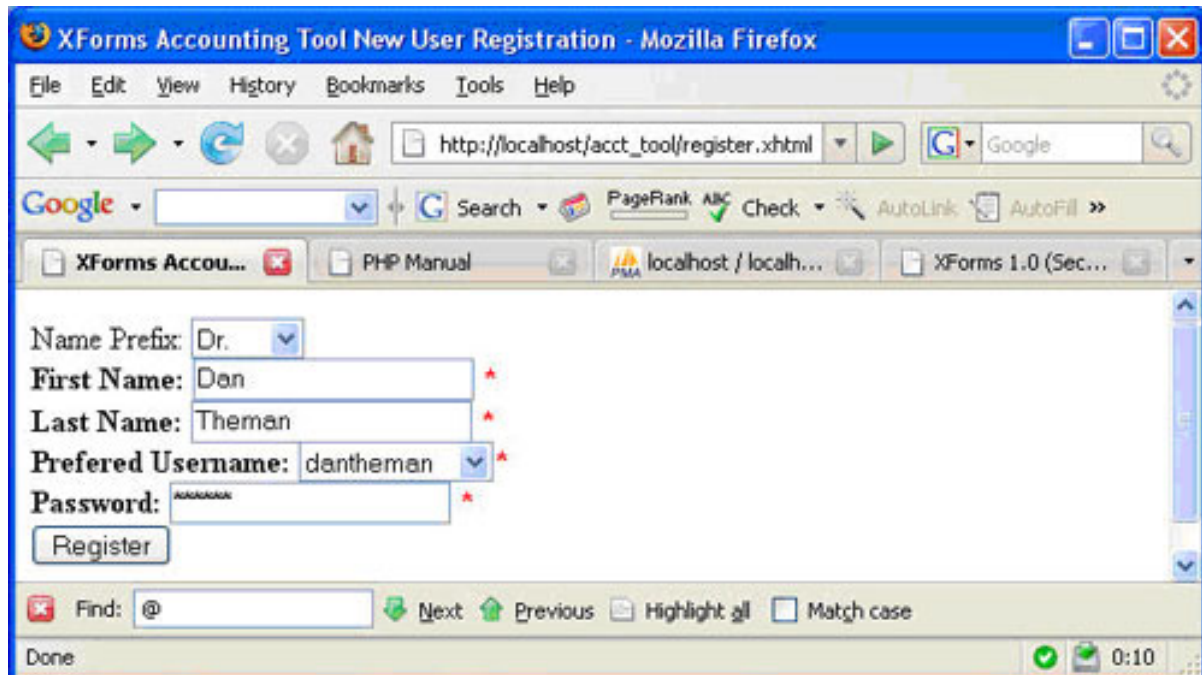
Most of the PHP should look familiar. The two significant differences are shown in bold. The first is a simple change to the query string used in the login script. This time, instead of asking if there is an entry with username and password that match, the goal is to discover if a username is taken or not. The "*" coupled with no mention at all of the password will return all the entries in the database that share the specified username. The user is instructed to select a different username if there are any results returned.

The next PHP to notice is the query that inserts data into the table. The syntax is trivial, and later parts of this series will monopolize on the simplicity and regularity of that syntax.

The last bolded section of code performs the "whisking" of the user back to the login page. Note the inclusion of "exit" in the bolded code. Exit is not a requirement of the redirection, but no more text may be displayed. If the PHP echoes anything prior to

calling "header," the call to "header" will result in an error message.

Figure 10. The completed registration form



Section 5. Summary

The login XForms form introduced a basic working subset of XForms concepts. Among the concepts introduced were some basic form controls, "input," "secret," "submit," and "trigger." The XForms control "output," which is used to display data that the user cannot directly alter, was also used. The concept of submission was demonstrated with the data used to replace the instance data behind the XForms implementation. That activity enabled the use of relevant to hide the form and the presence of the new data enabled the demonstration of "repeat" on a nodeset to create a menu.

Future installments of this series refine both the XForms implementations as necessary and the skills involved in XForms implementation. Next, Part 3 will use the concept of asset management to demonstrate more in-depth usage of repeats and another variation of the load action and more.

Downloads

Description	Name	Size	Download method
Sample code for this tutorial	accttool_part2_code.zip	8KB	HTTP

[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).
- Get the [The PHP Manual](#).
- Learn more about XForms submission events in [XForms tip: Using form submission events](#) (developerWorks, November 2006).
- Find out how to accept XForms data in [Java](#) (developerWorks, October 2006), [Perl](#) (developerWorks, October 2006), and [PHP](#) (developerWorks, October 2006).
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- Learn all about XML and more about XForms in the IBM developerWorks [XML zone](#).

Get products and technologies

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

Discuss

- [Participate in the discussion forum for this content](#).
- [developerWorks blogs](#): Get involved in the developerWorks community.

About the author

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.