Computer Science E-75: Building Dynamic Websites
Harvard Extension School
Spring 2008

# Section 2, Continued: PHP, Continued

Since we didn't have lecture this Monday, this week's notes will focus on a couple general features of php.

## Another XML Example: staff.xml

```
<staff>
      <professors>
            <professor name="David">
                  <color>Gold</color>
            </professor>
      </professors>
      <tfs>
            <tf name="Dan">
                  <color>Blue</color>
            </tf>
            <tf name="Daniel">
                  <color>Green</color>
            </tf>
            <tf name="Chris">
                  <color>Red</color>
            </tf>
            <tf name="John">
                  <color>Orange</color>
            </tf>
            <tf name="Keito">
                  <color>Yellow</color>
            </tf>
      </tfs>
</staff>
```

## Loading staff.xml: Using SimpleXML

SimpleXML is a PHP extension (available in PHP 5+) that allows users to easily manipulate and/or use XML data.

First, load the xml file (this can be done in two ways):

1. Initialize a new SimpleXMLElement object with a string containing the XML document as its sole parameter:

```
$xml = new SimpleXMLElement(file_get_contents('staff.xml'));
```

2. Use function simplexml_load_file() with the name of the xml file (staff.xml) as its argument:

```
$xml = simplexml_load_file('staff.xml');
```

It may prove useful to check if the file exists before executing this line (error-checking is a good practice in general). This can be done using a simple if conditional:

```
if (file_exists('staff.xml'))
     $xml = . . .;
```

If you'd like to see the format of the object $xml, you can print it using the function print_r() as follows (the first print makes sure everything is indented nicely):

```
print('<pre>');
print_r($xml);
```

**Accessing Elements of staff.xml: Iterating and Using Xpath**

Iterating through the xml file:

To access different parts of your xml file, the `->` (arrow) operator comes in handy. The arrow operator accesses child elements (for example, in staff.xml, professor is a child of staff). Syntax for this is as follows:

```
$xml->professors
```

You can access children of children using the arrow operator as well.

```
$xml->tfs->tf[0]
```

Putting the [0] after tf causes you to access the first tf (the one corresponding with Dan). From there, you can access attributes:

```
echo $xml->tfs->tf[0]['name'];
```

Prints out: `Dan`

You can also print out information stored between tags.

```
echo $xml->tfs->tf[0]->color;
```

Prints out: `Blue`

For information about **foreach loops**, consult last week's notes.

Using Xpath:

Xpath can be used to query XML documents and immediately jump to certain areas, without having to iterate over the entire document. Slashes ("/") are typically used to indicate children, and the @ symbol is used for attributes.

To refer to the portion of staff.xml having to do with tf, Dan, the following path would be used:

```
/staff/tfs/tf[@name="Dan"]
```

Then, the following code would be used to access this path:

```
$dan = $xml->xpath('/staff/tfs/tf[@name="Dan"]');
```

To use `$dan`, then, we can print out Dan's color:

```
echo $dan[0]->color;
```

Which prints: `Blue`

The foreach loop can also be used with Xpath, consult last weeks notes for the syntax.

**Note:** If you want to check whether an attribute name is equal to a variable (in other words, [@name="$input"]), the variable declaration must be escaped from the quotes (lest it check whether the attribute name is "$input" and not "value_of_input"). Ways of fixing this are as follows (using sushi implementation found in last week's section notes:

```
$xml->xpath("/sushi/fish
[@name='$squid']/variations/variation");

$xml->xpath("/sushi/fish[@name=\"$squid\"]/variations/variation");

$xml->xpath("/sushi/fish/
[@name='" . $squid ."']/variations/variation");
```

**Taking and Processing Form Input:**

Form declarations:
Forms and their declarations should be familiar to you by now, but if not, no need to fret. Here is a typical declaration of a form:

```
<form action="input.php" method="get">
     <input type="text" name="input1" />
     <input type="submit" value="Enter" />
</form>
```

Form's attribute, action, specifies to what page the information should be passed. Its method describes how it should be passed (get, post, etc.).

Inputs types can vary, depending on their usage. Some common types are text (for normal text), hidden (for a value that isn't shown), password (for input to be *'d), and submit (the submit button that sends the information).

Using user form input:

The page to which information is passed can then manipulate the information. Say, for example, that input1 was the amount of a bill at a restaurant, and that we want to calculate and display a 15% tip on the next page. The page input.php's code would be as follows.

```
<?php
      $bill = $_GET['input1'];
      $tip = $bill * .15;
      echo $tip;
?>
```

We use the $_GET[] because the information was passed to input.php by method GET. From there, we can perform mathematical calculations on $bill.

**Validate User Input Server-Side:**

PHP supports several functions that are useful to validate user input. These include:

- trim();
- preg_match();
- is_int();
- and our friend, ==

All of these functions are explained in great detail on php.net. A short description of each follows.

trim($str);

- Trim returns the string $str with all whitespace at the beginning and end of $str removed. Adding another argument can cause specific characters to be removed.

preg_match();

- Preg_match performs a regular expression match (whether two expression are the same. There are a lot of great things you can do with preg_match, including segregating specific important parts of a user input

is_numeric();

- Variations of this function are available for doubles, ints, and the like, but this function simply checks whether its argument is a numeric value.

==

- The double equal sign (==) is used in conditional statements to check whether one value (string or otherwise) is equal to another. Remember, there is a triple equal sign (===) in php as well, which tests whether two pieces of data are of the same value **and** type.

Please consult php.net for further documentation and examples of each of these and similar functions.

**Storing Objects or Associative Arrays in $_SESSION:**
The session superglobal is very useful, in that the information stored within it can be accessed by every page with a session_start() declaration at the top. For more information on sessions, please consult last week's section notes.

Virtually anything can be stored in the session variable. $_SESSION is basically an associative array where $_SESSION['value'] corresponds to certain information stored with the index 'value'.

Here's some code from the shopping cart example (from last week's notes) that shows you how to store an associative array in $_SESSION.

```
<?php
//Start the session
session_start();

//Read in variables from $_POST
$price     = $_POST['price'];
$item      = $_POST['item'];
$quantity  = $_POST['quantity'];

//We define an associative array with the details of our new item
$array = array(
    'item' => $item,
    'unitprice' => $price,
    'quantity' => $quantity
);

//If $_SESSION['cart'] hasn't yet been defined, define it as an ar
ray
if (!isset($_SESSION['cart']))
    $_SESSION['cart'] = array();

//Append the item to the $_SESSION['cart'] session variable
// ($array[] means "append a new item to the end of $array")
$_SESSION['cart'][] = $array;

?>
```

$array is first defined as an associative array using the array() function in php. $_SESSION['cart'] is then defined to be an array, using the same array() function. $array is then passed to the $_SESSION['cart'] variable with the line,

```
$_SESSION['cart'][] = $array;
```

Now you have an associative array stored in $_SESSION['cart']!