

## Exercise 1. RSS Aggregation application - A quick orientation

This series of exercises will result in a fully functional RSS reader with the ability to subscribe to and tag RSS feeds. We will work on this exercise for the remainder of the semester. The files for all of these exercises are located at: <http://www.ils.unc.edu/~mitcheet/feeds/>. For these exercises, you can either type the code in by hand or download portions of the code from this site. Note that the PHP files are also saved as text files (eg ex3.txt). Each exercise has a corresponding PHP file (eg ex1.php, ex2.php) and where necessary XSL and JavaScript files. If you have questions about a specific function in these exercises, a good reference resource is: <http://www.gotapi.com/jsdomw3s>.

Take a look at the finished application at <http://www.ils.unc.edu/~mitcheet/feeds/ex11.php>

In our previous exercises, we worked with PHP, XSL, and CSS to read and display RSS feeds. We will begin with some of the tools from these first few exercises and create a basic page framework to work with. This application uses XML data, XSL transformation stylesheets, CSS, PHP, JavaScript, and Ajax to create an application that will allow you to subscribe to and view RSS feeds. The primary function of the application is to allow users to subscribe to and view RSS feeds. Features of the application include: the ability (via cookies) to allow the creation of multiple accounts, the ability to subscribe to and unsubscribe from RSS feeds, and the ability to view feeds in the application and link to specific stories. The application uses CSS, JavaScript, and Ajax to enhance the user experience and degrades gracefully over time.

In our first few exercises we will work with the basics of PHP, XSL, and CSS. In exercise 6 we will add database functionality to our application using MySQL. Exercise 7 add subscription/unsubscription functionality. In exercise 10 we add login/logout functionality and finally in exercise 11 we introduce Ajax.

Exercise 1.	A quick orientation .....	1
Exercise 2.	Create the HTML page framework.....	2
Exercise 3.	Create an XML configuration file and transformation function.....	3
Exercise 4.	Showing our feeds .....	7
Exercise 5.	Combine the two XSL files using named templates and parameters .....	10
Exercise 6.	Migrating our feed-list to a relational database .....	13
Exercise 7.	Enabling feed subscription and un-subscription.....	18
Exercise 8.	Creating a PHP functions file .....	24
Exercise 9.	Use JavaScript to control form display.....	29
Exercise 10.	Adding a Login function to our page.....	32
Exercise 11.	Introducing Ajax.....	35

## Exercise 2. Create the HTML page framework

### 2.1. Create your stock HTML page with a reference to an external stylesheet

To save time, you can go to <http://www.ils.unc.edu/~mitcheet/feeds/> and download the files for exercise 2.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>RSS Feed Aggregator </title>
    <link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
    <meta name="DC.title" content="RSS Feed reader"/>
    <meta name="DC.author" content=" "/>
    <link rel="stylesheet" type="text/css" href="/main.css"/>
  </head>
  <body>
    <div id="header">
      <h1>RSS aggregation service</h1>
    </div>
    <div id="nav"></div>
    <div id="main"></div>
  </body>
</html>
```

### 2.2. Create our stock CSS file

```
/*INLS572 External CSS File */
*{margin:0; padding:0;}
html {
  font: small/1.4 "Lucida Grande", Tahoma, sans-serif;
}
body {
  font-size: 92%;
}
/*setup our main body entry */
#main {float:left; padding: 2em 2em 2em 2em;}
#main p {margin:1em; }

/*format our header*/
#header {background-color:#FFFFCC; padding:0; margin:0; height:8em;}

/*style our links*/
a {
  text-decoration: none;
  display: block; /* to increase clickable area as a's
  default to inline */
  color:#000000;
  text-decoration: none;
  padding: 0 15px;
  line-height: 2.5;
  border-bottom: 1px solid #FFF;
}
```

## Exercise 3. Create an XML configuration file and transformation function

In this exercise, we will create our own RSS based XML file to serve as our program configuration file. This file will store all of our subscribed RSS feeds. We will also create two PHP functions to transform this XML data for our application. In this exercise we will be using a set of DOMXML functions in PHP. The documentation for these functions are available here - <http://us.php.net/manual/en/function.domxml-open-file.php>, and <http://us.php.net/manual/en/function.domxml-xslt-stylesheet.php>.

### 3.1. Create the XML configuration file.

Create the file (use your own feeds if you like) and save it with a .xml extension. We will use this file for the remainder of this exercise

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="ex5navi.xsl"?>
<rss>
  <channel>
    <item>
      <title>NYT Health</title>
      <link>http://www.nytimes.com/services/xml/rss/nyt/Research.xml</link>
    </item>
    <item>
      <title>CNN Entertainment</title>
      <link>http://rss.cnn.com/rss/cnn_showbiz.rss</link>
    </item>
    <item>More feeds here.....</item>
    <item>
      <title>Flickr current images</title>
      <link>http://api.flickr.com/services/feeds/photos_public.gne?format=rss2</link>
    </item>
  </channel>
</rss>
```

### 3.2. Create the XSL parser in PHP

Create a PHP function that transforms our XML file using an XSL template. Add the following function to your stock page, save the page with a .php extension. This function can be added anywhere to your page but I would suggest the body div.

```
<?php
function translateXML ($xmlinput, $xslfile, $params) {
  $xmlfile = domxml_open_file($xmlinput);
  $xslt = domxml_xslt_stylesheet_file($xslfile);
  $htmloutput = $xslt->process($xmlfile, $params);
```

```

    echo $htmloutput->dump_mem();
}
?>

```

Meaning of the above function:

- function
  - The PHP command to create a new function
- translateXML
  - Our function name, we could call this anything
- (\$xmlinput, \$xslfile, \$params)
  - The variables that we will pass this function. \$xmlinput is our xml file to transform, \$xslfile is our xsl template, \$params is an associative array of parameters that we can pass to the xsl file. For now, our \$params variable is empty.
- \$xmlfile = domxml\_open\_file(\$xmlinput);
  - This command creates a new variable and assigns it the value of the output of the function domxml\_open\_file(\$xmlinput). Domxml\_open\_file is a built-in function for PHP that opens an XML file and reads its content. \$xmlinput is a variable that points to the location of our xml configuration file created in step 1 of this exercise
- \$xslt = domxml\_xslt\_stylesheet\_file(\$xslfile);
  - This function does for the stylesheet what the previous command did for the xml document. It opens the stylesheet and creates a variable (really an object) for processing
- \$htmloutput = \$xslt->process(\$xmlfile, \$params);
  - This function creates a variable \$htmloutput that has the value of the \$xslt->process() function. In this case, we pass our xmlfile and parameters to the process and receive transformed content back.
- echo \$htmloutput->dump\_mem();
  - This function prints the output of the variable \$htmloutput to the screen. The dump\_mem() function is part of the suite of XSL functions we are using.

### 3.3. Create an XSL file to transform our XML file for navigation.

This XSL file will access the data in our XML configuration file and output an unordered list of feed links. We will go ahead and create a dynamic link in our href entries by inserting values for pageProcess and feed. You can see this happening in the linkLocation variable assignment. Enter this file and save it as navigation.xsl (or something similar)

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <!--Save some time, create a variable with your page name so you can easily create URLs to your page -->
    <xsl:variable name="applicationPath" select="http://www.ils.unc.edu/~mitcheet/feeds/ex2.php"/>
    <!-- create an unordered list to hold your menu items -->
    <ul class="navigation">Subscribed Feeds
    <xsl:for-each select="/rss/channel/item">
      <!--We want to create a variable called linkLocation that combines our applicationPath variable with a dynamic
      querystring value (in this case a pageProcess and a showFeed variable) that we can use to control page functionality
      later -->
      <xsl:variable name="linkLocation" select="concat($applicationPath, '?pageProcess=showFeed&feed=', ./title)"/>
      <!--output things-->
      <li><a href="{ $linkLocation }"><xsl:value-of select="./title"/></a></li>
    </xsl:for-each>
    </ul>
  </xsl:template>

```

```
</xsl:stylesheet>
```

- The above stylesheet uses the concepts we have already learned (templates, xpath, and variables), and just uses these concepts to create an unordered list of elements to style.
- One change is the creation of an applicationPath variable. We will use this to control our page by loading the RSS feed from our navigation menu. This functionality requires two components an applicationPath variable and a dynamic querystring variable
  - `<xsl:variable name="applicationPath" select="http://www.ils.unc.edu/~mitcheet/feeds/ex2.php"/>`
    - Set this to the location of your rss.php page
  - `<xsl:variable name="linkLocation" select="concat($applicationPath, '?pageProcess=showFeed&feed=', ./title)"/>`
    - Notice where this linkLocation variable is created. We add a querystring of `?pageProcess=showFeed&feed=', ./title`. This creates a link for each element that we output that will tell the program to do something specific

### 3.4. *Modify the navigation div to call our new function*

Modify the navigation div of our file to call our new translateXML function. Notice how we pass the values of our xml file, our xsl file and (for now) this empty parameter variable.

```
<div id="nav"><?php translateXML("./feeds.xml", "./navigation.xsl", $params)?></div>
```

### 3.5. *Add elements of style to our CSS page for the navigation*

```
/*Content from exercise 2 */
/*Create some basic formatting for navigation */
#nav {
  margin: 0;
  padding: 0;
  background-color:#FFFFCC;
  list-style-type: none;
  width: 210px;
  height:600px;
  float: left; /* Contain floated list items */
}
#nav li {
  margin: 0;
  padding: 0;
  float: left; /* This corrects the */
  width: 100%; /* IE whitespace bug */
font-size: 12px;
}
#nav ul {
font-size:14px;
```

```
font-weight: bold;
}
/*Add a hover style to our navigation */
#nav a:hover {
  background: #4F4532;
}
```

## Exercise 4. Showing our feeds

In order to have the PHP page show a feed when we click on the link in the navigation menu, we have to add some decision making processes PHP page and an XSL stylesheet to format the feed. Fortunately, because we have already created the function for parsing XML data we do not have to worry about that process. In order to accomplish this we will have to pass a feed name to our PHP page and have the PHP page call the XSL transformation template and pass the feed name to that template so it will only show data from that feed. We will accomplish the first task using our querystring and the second task using the parameter variable for our XSL function.

### 4.1. Create an XSL stylesheet that will output RSS feed data in an unordered list

We completed this exercise in prior classes. An example output stylesheet is below. If you test this stylesheet, you should see every feed that is in your XML configuration file output to the screen

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <!--Do an inefficient match on any title elements -->
    <xsl:for-each select="//title">
      <div class="rssfeed">
        <!-- create a variable for our feed link so we can easily insert it into our href-->
        <xsl:variable name="feedUrl" select="../link"/>
        <!-- we have to use the concat function with our version of PHP to get the document function to work correctly -->
        <h1><a href="{ $feedUrl }"><xsl:value-of select="document(concat($feedUrl, '))/rss/channel/title/text()"/></a></h1>
        <ul>
          <!-- basically repeat the process for each item in the feed -->
          <xsl:for-each select="document(concat($feedUrl, '))/rss/channel/item">
            <xsl:sort select="title"/>
            <xsl:variable name="itemUrl"><xsl:value-of select="./guid"/></xsl:variable>
            <li><a href="{ $itemUrl }"><xsl:value-of select="./title"/></a></li>
          </xsl:for-each>
        </ul>
      </div>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

### 4.2. Add a call to translateXML for our feed display

Now, add a call to the translateXML function in our PHP page at the place where we want our feed listings to go. This should be in our main div so that the content will show up here.

```
//This is part of the PHP script we created earlier with an added call to translateXML
<div id="main">
  <?php
    translateXML("./feeds.xml", "./showfeed_all.xml", $params);

    function translateXML ($xmlinput, $xslfile, $params) {
      $xmlfile = domxml_open_file($xmlinput);
```

```

    $xslt = domxml_xslt_stylesheet_file($xslfile);
    $htmloutput = $xslt->process($xmlfile, $params);
    echo $htmloutput->dump_mem();
  }?>
</div>

```

Try running this version of the program and see how the output looks. Once you have these components working, we will move on to using our navigation links to limit which feeds get shown.

### 4.3. *Modify our navigation links to show a specific feed*

This process requires two steps. First, we need to add the ability to pass values to our XSL file from the link on the page. Second, we need to modify our XSL file to make decisions about which feed to show. We can modify our PHP script to pass parameters to the XSL file by populating the parameters array referred to in exercise 2 with variables from the querystring:

#### 4.3.1. **Giving our PHP page the ability to send and receive querystring information**

- 1) In the previous part of this exercise, we gave our stylesheet the ability to output dynamic links for navigation. We now have to build the ability to read and process that information in our PHP page. We will do this by passing an associative array to our XSL file through PHP. If we create a \$params array with two values (feedName and pageProcess) just before our function call in the main body of our PHP page, we will be able to pass those values to our XSL file. The params array is built using values from the \$\_REQUEST variable which we will populate with our navigation links.

```

<div id="main">
  <?php
    //Defining our Params array
    $params = array('feedName' => $_REQUEST['feed'], 'pageProcess' => $_REQUEST['pageProcess']);
    translateXML("./feeds.xml", "./showfeed_all.xsl", $params);
    .....(see step 2 for the rest of the "main" div)

```

- 2) The \$params line creates an associative array (reference by name not by number) and associates the value from our \$\_REQUEST command (PHP's way of accessing the querystring data) with values to pass to our XSL file. In this case, we are passing two array values, one for feedName (the name of our RSS feed) and pageProcess (a command telling our page which process to follow).
- 3) In order to add decision making capacity to our XSL file, we need to reference the parameters passed to it and then make a decision based on the value of the parameters. Create a new stylesheet called showfeed.xsl with the following code. You can name the stylesheet whatever you want but be sure that the name is reflected in the transformXML call in the PHP script.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <!-- tell our template about the two parameters it is getting passed -->
  <xsl:param name="feedName"/>
  <xsl:param name="pageProcess"/>

  <!-- Our main processing template-->
  <xsl:template match="/">
    <!--Save some time, create a variable with your page name so you can easily create URLs to your page -->

```

```

<xsl:variable name="applicationPath" select="http://www.ils.unc.edu/~mitcheet/feeds/ex3.php"/>
<!--Since we really are only doing one thing here we don't really need a choose but we will be adding additional functionality here later☺ -->
  <xsl:choose>
<!-- Examine the parameter that was passed to our stylesheet and call a specific template -->
    <xsl:when test="$pageProcess = 'showFeed'">
      <xsl:call-template name="showFeed"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<!-- our second template (showFeed) The rest of this template should look familiar with the exception of the if statement below-->
<xsl:template name="showFeed">
  <xsl:for-each select="//title">
    <!-- We add a test to see if our current title is for the RSS feed we are looking for. If so, we go into a new loop and output all of the items for that feed -->
    <xsl:if test=". = $feedName">
      <div class="rssfeed">
        <xsl:variable name="feedUrl" select="../link"/>
        <h1><a href="{ $feedUrl }"><xsl:value-of select="document(concat($feedUrl, '))/rss/channel/title/text()"/></a></h1>
        <ul>
          <xsl:for-each select="document(concat($feedUrl, '))/rss/channel/item">
            <xsl:sort select="title"/>
            <xsl:variable name="itemUrl"><xsl:value-of select="../guid"/></xsl:variable>
            <li><a href="{ $itemUrl }"><xsl:value-of select="title"/></a></li>
          </xsl:for-each>
        </ul>
      </div>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

- 4) Finally, we need to style our data a bit. A simple stylesheet addition is shown below. You can add this you're the main stylesheet

/\*Elements from exercise 3\*/

```

.rssfeed div { width:400px; display: block;}
.rssfeed ul {list-style-type:none;}
.rssfeed h1 {background-color:#FFF; padding: 0 0 0 0; margin: 0; border-bottom-color:#000000; border-bottom:solid;}
.rssfeed { margin:0px; padding: 0 0 0 0;}
.rssfeed span {
display: block;
}

```

## Exercise 5. Combine the two XSL files using named templates and parameters

In this exercise, we will combine our two XSL files into a single file using decision making to determine which template to access. Combining the files allows us to simplify our file structure and benefit from code re-use. In order to accomplish this, we need to combine the two XSL files you created in exercises 2 and 3, and add parameters to the navigation feed (so that the combined files know how to transform data).

### 5.1. Combine the two XSL files.

The new new page structure will look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="feedName"/>
  <xsl:param name="pageProcess"/>
  <xsl:variable name="applicationPath" select="http://www.ils.unc.edu/~mitcheet/feeds/ex3.php"/>
  <xsl:template match="/">
    Main template content.....
  </xsl:template>

  <xsl:template name="showFeed">
    Showfeed template content.....
  </xsl:template>

  <xsl:template name="showNav">
    Show Nav template content.....
  </xsl:template>
</xsl:stylesheet>
```

Notes:

Our XSL:template match="/" is our main processing template. Notice how we define our parameters at the stylesheet level. This allows us to reference these as "global" variables. We create two other templates (showFeed and showNav) which we need to put the contents from other two files in.

### 5.2. Adding Logic:

In our main template ("/"), we just need a choose statement to help the program know when/what to do. Add the following code to your main template and make sure that your named template calls match the template names you are calling in the XSL file. Notice also how we use the test attribute of our <xsl:when> element to see what the value of our passed parameter \$pageProcess is equal to. Defining paramters, variables, and system variables (like pageProcess) can be a confusing part of program design.

```
<xsl:choose>
  <xsl:when test="$pageProcess = 'showFeed'">
    <xsl:call-template name="showFeed"/>
  </xsl:when>
  <xsl:when test="$pageProcess = 'showNav'">
    <xsl:call-template name="showNav"/>
  </xsl:when>
  <xsl:otherwise>
```

```

    </xsl:otherwise>
</xsl:choose>

```

The complete XSL transformation file should look like:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:param name="feedName"/>
  <xsl:param name="pageProcess"/>
  <!-- add a feed URL param -->
  <xsl:param name="feedUrl"/>
  <!-- Be sure your page name is correct -->
  <xsl:variable name="applicationPath" select="http://www.ils.unc.edu/~mitcheet/feeds/ex4.php"/>
  <xsl:template match="/">

    <xsl:choose>
    <xsl:when test="$pageProcess = 'showFeed'">
      <xsl:call-template name="showFeed"/>
    </xsl:when>
    <xsl:when test="$pageProcess = 'showNav'">
      <xsl:call-template name="showNav"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>

  <xsl:template name="showFeed">
  <xsl:for-each select="//title">
    <xsl:if test=". = $feedName">
      <div class="rssfeed">
        <xsl:variable name="feedUrl" select="../link"/>
        <h1>
          <a href="{ $feedUrl }"><xsl:value-of select="document(concat($feedUrl,))/rss/channel/title/text()"/></a></h1>
        <ul>
          <xsl:for-each select="document(concat($feedUrl,))/rss/channel/item">
            <xsl:sort select="title"/>
            <xsl:variable name="itemUrl"><xsl:value-of select="./guid"/></xsl:variable>
            <li><a href="{ $itemUrl }"><xsl:value-of select="./title"/></a></li>
          </xsl:for-each>
        </ul>
      </div>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

  <xsl:template name="showNav">
  <ul class="navigation">Subscribed Feeds
  <xsl:for-each select="/rss/channel/item">
    <xsl:variable name="linkLocation" select="concat($applicationPath, '?pageProcess=showFeed&feed=', ./title)"/>
    <li><a href="{ $linkLocation }"><xsl:value-of select="./title"/></a></li>
  </xsl:for-each>
  </ul>
</xsl:template>
</xsl:stylesheet>

```

### 5.3. Modify our transformXML function call in our main PHP page

Modify the transformXML call in the navigation div of your PHP page to pass the showNav value for the associative array location page process:

```
<div id="nav"><?php  
$params = array('pageProcess' => 'showNav');  
translateXML("./feeds.xml", "./templates.xsl", $params);  
?></div>
```

## Exercise 6. Migrating our feed-list to a relational database

In this exercise, we will abandon the XML configuration file created in exercise 3 and begin using a table from a MySQL database to drive our page. This is a shared table for the entire class so we will also have to add some hidden form elements (a username) to make sure that we only see our data on our page. While our navigation functionality will no longer use the xml file, our showFeed function will continue to be used.

### 6.1. A quick overview of our database

Our MySQL database consists of one table called feeds:

feeds			
id	username	feedname	feedURL
1	mitcheet	NYT - Health	http://www.nytimes.com/services/xml/rss/nyt/Research.xml
2	mitcheet	Rolling Stone - New Music	http://www.rollingstone.com/blogs/breaking/rss.xml
3	mitcheet	BBC - Breaking News	http://newsrss.bbc.co.uk/rss/newsonline_world_edition/front_page/rss.xml
4	mitcheet	Flickr - Current Images	http://api.flickr.com/services/feeds/photos_public.gne?format=rss2
8	mitcheet	Real Estate	http://www.nytimes.com/services/xml/rss/nyt/RealEstate.xml
11	mitcheet	NYT - Thursday Styles	http://www.nytimes.com/services/xml/rss/nyt/ThursdayStyles.xml

As you can see from the table, it contains an automatic number column (id), a username column, a feedname column, and a feedURL column. This should look similar to our XML configuration file from the last few exercises. Because we will all be using the same table, we needed to add in a username field and in order to make sure that we have data integrity when we start inserting and deleting values we will use the primary identifier (id) to complete those tasks.

### 6.2. Add some feeds to the database for your username

In order to have some data to work with, you need to subscribe to a few feeds. Pick a username to work with for the remainder of these exercises and subscribe to some feeds:

- 1) In order to add some content to our database, you can go to <http://ils.unc.edu/~mitcheet/rssreader/adddata.php>.

Some content entry guidelines are:

- a. The URL must point to a valid RSS feed
- b. The FeedName can be whatever you wish
- c. The username must be your username (one that you will use on your page)
- d. Add four or five feeds to the database for your name.

### 6.3. Modify our navigation function to use the database

In order to get data from our database on to our page, we need to use some MySQL functions with PHP. The set of MySQL functions in PHP is not particularly complicated and consists of the steps of initialize, connect, query, retrieve, and close. In order to use the MySQL functions you simply need access to a MySQL database.

Here is a sample function (copied from the navigation function we will be using:

```
function navigation () {
    $server = "http://www.ils.unc.edu/~mitcheet/feeds/ex5.php";
    $connection = mysql_connect("pearl.ils.unc.edu", "inls572_spring08", "yreuq572");
    mysql_select_db("inls572_spring08", $connection);
```

```

$result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);
while ($row = mysql_fetch_array($result, MYSQL_NUM))
{
    $output .= $row[2] //content from database
}
echo $output;
mysql_close($connection);
}

```

Notice the specific functions used to create the connection:

mysql\_connect() - creates a connection to the database  
mysql\_select\_db - selects a database to talk to  
mysql\_query - executes a query on the database  
mysql\_fetch\_array - retrieves the results as a regular array  
mysql\_close - closes the connection.

Once we define our navigation function, we can simply reference it in our navigation div.

## 6.4. *Creating a new showFeed function based on our database*

Because our navigation links will return a unique id number instead of a feedURL or title, we will have to connect to our database in order to retrieve the URL. Once we retrieve the URL, we will be able to call the XSL function we created earlier to retrieve the feed and output it as HTML. The new PHP showFeed function is below. We first connect to our database in order to get the feed URL for a given id and then pass this URL to our XSL transformation function:

```

function showFeed () {
    $server = "http://www.ils.unc.edu".$PHP_SELF;
    $connection = mysql_connect("pearl.ils.unc.edu", "inls572_spring08", "yreuq572");
    mysql_select_db("inls572_spring08", $connection);
    //The SQL statement is different for this function
    $result = mysql_query("SELECT * from feeds where id = ".$_REQUEST['feed'], $connection);
    while ($row = mysql_fetch_array($result, MYSQL_NUM))
    {
        //For this function, we create the params array and call our translateXML function
        //Make sure your XSL call matches your XSL file below (in bold)
        $params = array('feedUrl' => $row[3], 'pageProcess' => $_REQUEST['pageProcess']);
        translateXML ($row[3], './templates.xml', $params);
    }
}

```

## 6.5. *Modify our XSL for the new showFeed function*

Now that we have a new showFeed function, we need to change our showFeed XSL template. Rather than modifying our current showFeed XSL template, lets copy it and make the following modifications:

```

<!--Call it something different-->
<xsl:template name="MYSQL_showFeed">
    <!-- Remove the for-each that looks at //title and if test that checks the value of feed name-->
    <div class="rssfeed">
        <!-- Remove the variable definition for feed URL, we will use the parameter passing process instead -->
        <h1>
            <a href="{ $feedUrl }"><xsl:value-of select="document(concat($feedUrl, '))/rss/channel/title/text()"/></a></h1>

```

```

    <ul>
      <xsl:for-each select="document(concat($feedUrl,"))/rss/channel/item">
        <xsl:sort select="title"/>
        <xsl:variable name="itemUrl"><xsl:value-of select="./guid"/></xsl:variable>
        <li><a href="{ $itemUrl } "><xsl:value-of select="./title"/></a></li>
      </xsl:for-each>
    </ul>
  </div>
<!--remember to remove your for-each and if closing elements-->
</xsl:template>

```

Now make sure that we reference our feedURL parameter in the top of the XSL document:

```

<!-- add a feed URL param -->
<xsl:param name="feedUrl"/>

```

Finally, modify your `<xsl:choose>` element to call the new `showFeed` function when necessary. Add the following `<xsl:when>` to your choose:

```

<xsl:when test="$pageProcess = 'MYSQL_showFeed'">
  <xsl:call-template name="MYSQL_showFeed"/>
</xsl:when>

```

## 6.6. Pulling it all together

This is a rather complex re-write of our PHP page. I would recommend copying the page from the last exercise and making the changes indicated in bold below. In essence, we are creating a new PHP function called `navigation` to replace our XSL navigation function and a new PHP function called `showFeed` that will take an id from the navigation URL in the webpage, connect to the database to retrieve the Feed URL, and pass that data to the XSL transformation file. Copy the PHP page you created in exercise 4 and make the following modifications (in bold):

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>RSS Feed Aggregator </title>
    <link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
    <meta name="DC.title" content="RSS Feed reader"/>
    <meta name="DC.author" content=" "/>
    <link rel="stylesheet" type="text/css" href="/main.css"/>
  </head>
  <body>
    <div id="header">
      <h1>RSS aggregation service</h1>
    </div>
    <!-- We will turn our navigation call into a database function. Call the function here and see code below for function contents -->
    <div id="nav"><?php navigation();?></div>
    <div id="main">
      <?php

```

```

/*We will substitute a simple If statement to check for a new page processs (MYSQL_showFeed) and we will call that function when we want to show feed contents. This function uses both our MySQL operations and our XSL transformation -*/

```

```

  if ($_REQUEST['pageProcess'] == "MYSQL_showFeed") showFeed();

```

```

function translateXML ($xmlinput, $xslfile, $params) {
  $xmlfile = domxml_open_file($xmlinput);
  $xslt = domxml_xslt_stylesheet_file($xslfile);

```

```

    $output = $xslt->process($xmlfile, $params);
    echo $output->dump_mem();

}
/*Here is our navigation function. This function connects to the MySQL server, completes a select statement to get data out
and then outputs that data to our webpage */

function navigation () {
/*To save some effort, we define a variable (called server) with our page path. The $PHP_SELF variable provides us with
our current page name */
    $server = "http://www.ils.unc.edu" . $_SERVER['PHP_SELF'];

//Create a connection to MYSQL
//In normal practice, we would not encode our username and password in this page, we would insert them as variables
//using an insert from a non web-accessible directory.
//Note the syntax mysql_connect(Server, Username, Password)
    $connection = mysql_connect("pearl.ils.unc.edu", "inls572_spring08", "yreuq572");

//Connect to our database
    mysql_select_db("inls572_spring08", $connection);

//Create a query to get data from the database, store the output in the $result variable
    $result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);

//Begin storing html in an variable that we can print out later - for now just open our Navigation UL
    $output = '<ul class="navigation">Subscribed Feeds';

//Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
while ($row = mysql_fetch_array($result, MYSQL_NUM))
{
//Write a mix of HTML and database content to the output variable
    $output .= '<li><a href="' . $server . '?pageProcess=MYSQL_showFeed&feed=' . $row[0] . '>' . $row[2] . '</a></li>';
}

//close our output variable
    $output .= '</ul>';

//echo your output variable to the browser
    echo $output;
}

/*he showFeed function uses both our MySQL connection and our transformXML function to get a reference to the RSS
feed and transform it to HTML. In good programming, you would not have two very similar functions, you would pull the
database functionality out into its own function */

function showFeed () {
    $server = "http://www.ils.unc.edu" . $_SERVER['PHP_SELF'];
    $connection = mysql_connect("pearl.ils.unc.edu", "inls572_spring08", "yreuq572");
    mysql_select_db("inls572_spring08", $connection);
    //The SQL statement is different for this function
    $result = mysql_query("SELECT * from feeds where id = " . $_REQUEST['feed'], $connection);
    while ($row = mysql_fetch_array($result, MYSQL_NUM))
    {
        //For this function, we create the params array and call our translateXML function
        $params = array('feedUrl' => $row[3], 'pageProcess' => $_REQUEST['pageProcess']);
        translateXML ($row[3], './templates.xml', $params);
    }
}

```

```
?>  
</div>  
  </body>  
</html>
```

## Exercise 7. Enabling feed subscription and un-subscription

Now that we have a database to work with, we will add some more interaction tools. In this exercise, we will add two forms to our page to enable feed subscription and un-subscription functionality. This will require elements of CSS, database interaction, and PHP.

### 7.1. Getting started

Lets start by copying our PHP file from the last exercise. In addition to adding the forms and functions to our PHP page, we will practice some good programming techniques using global variables and generalized functions that we can re-use. Rather than trying to break apart each section of the PHP script, this set of exercises will discuss what actions need to be taken and will rely on your debugging skills to review the complete PHP file at the end of this exercise for the changes that need to be made.

### 7.2. Create global variables

In order to create global variables, we just need to reference these variables at the top of our PHP file. In order to access these variables within specific functions however, we need to use the global command. See the following example:

```
<?php //Global Variables
    $server = "http://www.ils.unc.edu".$PHP_SELF;
    $dbserver = "pearl.ils.unc.edu";
    $username = "inls572_spring08";
    $pass = "yreuq572";
    $database = "inls572_spring08";

function sampleFunction() {
    global $server, $dbserver, $username;
}
?>
```

Go ahead and add a global variable section to the top of you PHP page. Create the global variables in the exercise above (keep all of the values as is). Notice the use of \$PHP\_SELF in the \$server variable. This is a convenient way of creating links that point back to the calling page and can be a big time-saver when creating URLs.

### 7.3. Add a subscription and unsubscription form to our navigation pane

Our subscription and unsubscription forms are not too complicated but require us to use a hidden form element (username) into which we insert the value of the username we used to create our database entreis. Notice how we reference our global variables in our form action sections. Also, notice our use of ID fields on our wrapping divs. We will use these ids later to use JavaScript to show and hide the forms. Change your nav div to match the code below:

```
<div id="nav">
    <?php navigation(); ?>
<!-- For the time being, lets hard-code our tasks in as members of an unordered list Notice our use of the hidden form element to limit which username is used to subscribe/unsubscribe -->
    <ul>Tasks
    <li>Subscribe to a feed
        <div id="subscriptionform">
```

```

<!--Notice how on the form action, we reference our global variable -->
    <form action="<?php global $server; echo $server;?>" method="post">
        <fieldset>
            <label for="feedURL">Feed URL</label><input type="text" name="feedUrl"/>
            <label for="feedName">Feed Name</label><input type="text" name="feedName"/>
<!--Put your own username in on the next line -->
            <input type="hidden" name="username" value="mitcheet"/>
            <input type="submit" name="pageProcess" value="Subscribe" />
        </fieldset>
    </form></div>
</li>
<li>Unsubscribe from a feed
    <div id="unsubscriptionform">
        <form action="<?php global $server; echo $server;?>" method="post">
            <fieldset>
                <input type="hidden" name="username" value="mitcheet"/>
                <select name="feedId">
<!-- We will use our MySQL functions to help create a list of feeds to unsubscribe from Call the function here, we will define it
below-->
                    <?php feedOptions(); ?>
                </select>
                <input type="submit" name="pageProcess" value="Unsubscribe" />
            </fieldset>
        </form>
    </div>
</li></ul></div>

```

## 7.4. Add the MySQL functions to process feed subscription and unsubscription

In this part of the exercise, we will create functions that complete insert and delete operations on our database. These three functions (insert data, delete data, populate drop-down menu for the unsubscribe form) all use our database connection functions. The main difference between them is the SQL statement that we use. Pay particular attention to the `$$SQL` variable in each one. If you are not familiar with SQL it is ok to simply copy and paste.

```

/* Our two new functions, subscribe and unsubscribe use the insert and delete functions of SQL but otherwise, look very similar to
our other two MySQL functions */
function subscribe () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
/* Notice the syntax of our Insert SQL statement - Insert into table (column list) values (value list); */
    $$SQL = "INSERT INTO feeds (username, feedname, feedURL) values ('".$_REQUEST['username'].',
    '".$_REQUEST['feedName'].', '".$_REQUEST['feedUrl'].')";
    $result = mysql_query($$SQL, $connection);
}
/* Here is our unsubscribe function. Notice the syntax of the delete statement delete from table where criteria = ‘*/
function unsubscribe () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
    $$SQL = "DELETE from feeds where id = '".$_REQUEST['feedId'].';
    $result = mysql_query($$SQL, $connection);
}
/*Our new function feedOptions()*/
function feedOptions() {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);

```

```

mysql_select_db($database, $connection);
//Create a query to get data from the database, store the output in the $result variable
$result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);
//Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
while ($row = mysql_fetch_array($result, MYSQL_NUM))
{
//Write a mix of HTML and database content to the output variable. In this case it is an option element for our form
$output .= '<option value="'. $row[0]."'>'. $row[2]. '</option>';
}
//echo your output variable to the browser
echo $output;
}

```

## 7.5. Tell the PHP page how to handle these new calls

Now that we have subscribe and unsubscribe functions, we need to let the page know when to call them. We will accomplish this by adding a switch statement to the beginning of our main div. Notice the use of the cases to check pageProcess below.

```

<div id="main">
<?php
/*We will substitute a simple If statement to check for a new page process (MYSQL_showFeed) and we will call that function
when we want to show feed contents. This function uses both our MySQL operations and our XSL transformation */
if ($_REQUEST['pageProcess'] == "MYSQL_showFeed") showFeed();
    global $server, $dbserver, $username, $pass, $database;
    switch ($pageProcess) {
        case "Subscribe":
            subscribe();
            break;
        case "Unsubscribe":
            unsubscribe();
            break;
    }
.....Remainder of Main Div

```

## 7.6. Modify the CSS for the page

As part of exercise 6, we also need to style our page a bit. We will use the ID fields placed on our forms to provide some general styling. Add the following content to our main.css file:

```

.subscribe {
    width:40%;
    display:inline;
    float:right;
}
label { display:block;font-weight: bold; }
fieldset { border:none;}

#subscriptionform {

    padding: 5px 5px 5px 5px;
    background-color:#CCFF99;
    border-style: solid;
    width:200px;
}

```

```
#unsubscribeform {
    padding: 5px 5px 5px 5px;
    background-color:#CCFF99;
    border-style: solid;
    width:200px;
}
```

## 7.7. Bringing it all together

In this exercise, we created two new forms, and three new functions in our PHP page, and added some CSS to our css file. While you should have been able to make these modifications based on the instructions above, the following code represents the entire PHP file as it should look at the end of exercise 7.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>RSS Feed Aggregator </title>
    <link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
    <meta name="DC.title" content="RSS Feed reader"/>
    <meta name="DC.author" content=""/>
    <link rel="stylesheet" type="text/css" href="/main.css"/>
  <!-- Lets go ahead and create some global variables to help simplify our functions. We will use these variables in all of our functions
  from here on out -->
  <?php //Global Variables
  $server = "http://www.ils.unc.edu".$_SERVER['PHP_SELF'];
  $dbserver = "pearl.ils.unc.edu";
  $username = "inls572_spring08";
  $pass = "yreuq572";
  $database = "inls572_spring08";
  ?>
  </head>
  <body>
    <div id="header">
      <h1>RSS aggregation service</h1>
    </div>
    <div id="nav">
      <?php navigation(); ?>
    <!-- For the time being, lets hard-code our tasks in as members of an unordered list Notice our use of the hidden form element to
    limit which username is used to subscribe/unsubscribe -->
    <ul>Tasks
    <li>Subscribe to a feed
      <div id="subscriptionform">
        <!--Notice how on the form action, we reference our global variable -->
        <form action="<?php global $server; echo $server;?" method="POST">
          <fieldset>
            <label for="feedURL">Feed URL</label><input type="text" name="feedUrl"/>
            <label for="feedName">Feed Name</label><input type="text" name="feedName"/>
        <!--Put your own username in on the next line -->
            <input type="hidden" name="username" value="mitcheet"/>
            <input type="submit" name="pageProcess" value="Subscribe" />
          </fieldset>
        </form></div>
      </li>
    <li>Unsubscribe from a feed
      <div id="unsubscribeform">
```

```

        <form action="<?php global $server; echo $server;?>" method="POST">
            <fieldset>
                <input type="hidden" name="username" value="mitcheet"/>
                <select name="feedId">
<!-- We will use our MySQL functions to help create a list of feeds to unsubscribe from Call the function here, we will define it
below-->
                    <?php feedOptions(); ?>
                </select>
                <input type="submit" name="pageProcess" value="Unsubscribe" />
            </fieldset>
        </form>
    </div>
</li></ul></div>
<div id="main">
<!-- In our Main processing area, we reference our global variables and use a switch statement to help guide our page actions. We
are now controlling three actions, showFeed, Subscribe, and unsubscribe. -->
    <?php
        global $server, $dbserver, $username, $pass, $database;
        switch ($pageProcess) {
            case "MYSQL_showFeed":
                showFeed();
                break;
            case "Subscribe":
                subscribe();
                break;
            case "Unsubscribe":
                unsubscribe();
                break;
        }
        function translateXML ($xmlinput, $xslfile, $params) {
            $xmlfile = domxml_open_file($xmlinput);
            $xslt = domxml_xslt_stylesheet_file($xslfile);
            $output = $xslt->process($xmlfile, $params);
            echo $output->dump_mem();
        }
    /*Our new function feedOptions()*/
    function feedOptions() {
        global $server, $dbserver, $username, $pass, $database;
        $connection = mysql_connect($dbserver, $username, $pass);
        mysql_select_db($database, $connection);
    //Create a query to get data from the database, store the output in the $result variable
        $result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);
    //Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
        while ($row = mysql_fetch_array($result, MYSQL_NUM))
        {
    //Write a mix of HTML and database content to the output variable. In this case it is an option element for our form
            $output .= '<option value="'. $row[0].'">'. $row[2]. '</option>';
        }
    //echo your output variable to the browser
        echo $output;
    }

    /* We have modified our navigation function just a bit to use the global variables. Otherwise, nothing changed here */
    function navigation () {
        global $server, $dbserver, $username, $pass, $database;
        $connection = mysql_connect($dbserver, $username, $pass);
        mysql_select_db($database, $connection);

```

```

//Create a query to get data from the database, store the output in the $result variable
$result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);
//Begin storing html in an variable that we can print out later - for now just open our Navigation UL
$output = '<ul class="navigation">Subscribed Feeds';
//Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
while ($row = mysql_fetch_array($result, MYSQL_NUM))
{
//Write a mix of HTML and database content to the output variable
$output .= '<li><a href="'. $server.'?pageProcess=MYSQL_showFeed&feed='.$row[0]."'>'. $row[2]. '</a></li>';
}
//close your output variable
$output .= '</ul>';
//echo your output variable to the browser
echo $output;
}

/* Same thing for showFeed. Only our use of global variables changed here */
function showFeed () {
global $server, $dbserver, $username, $pass, $database;
$connection = mysql_connect($dbserver, $username, $pass);
mysql_select_db($database, $connection);
//Create a query to get data from the database, store the output in the $result variable
$result = mysql_query("SELECT * from feeds where id = ".$_REQUEST['feed'], $connection);
//Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
while ($row = mysql_fetch_array($result, MYSQL_NUM))
{
$params = array('feedUrl' => $row[3], 'pageProcess' => $_REQUEST['pageProcess']);
translateXML ($row[3], './templates.xml', $params);
}
}

/* Our two new functions, subscribe and unsubscribe use the insert and delete functions of SQL but otherwise, look very similar to
our other two MySQL functions */
function subscribe () {
global $server, $dbserver, $username, $pass, $database;
$connection = mysql_connect($dbserver, $username, $pass);
mysql_select_db($database, $connection);
/* Notice the syntax of our Insert SQL statement - Insert into table (column list) values (value list); */
$sql = "INSERT INTO feeds (username, feedname, feedURL) values ('".$_REQUEST['username'].',
".$_REQUEST['feedName'].', '".$_REQUEST['feedUrl'].')";
$result = mysql_query($sql, $connection);
echo $result;
}

/* Here is our unsubscribe function. Notice the syntax of the delete statement delete from table where criteria = '*/
function unsubscribe () {
global $server, $dbserver, $username, $pass, $database;
$connection = mysql_connect($dbserver, $username, $pass);
mysql_select_db($database, $connection);
$sql = "DELETE from feeds where id = ".$_REQUEST['feedId'];
$result = mysql_query($sql, $connection);
echo $result;
}
?>
</div>
</body>
</html>

```

**Preview your changes and make sure that you can subscribe and unsubscribe to feeds.**

## Exercise 8. Creating a PHP functions file

Now that we have a lot of functions in our main processing area, it would be much easier to have our functions in a separate file. This will make our page simpler and give us the ability to re-use these functions on an as-needed basis. In this exercise we want to accomplish two things. First, we need to change the way the page works so that the subscribe and unsubscribe functionality work better. Two of the main issues with the functions as designed in exercise 7 is that they do not automatically update the navigation menu and they are susceptible to errors if the user reloads the page. In order to fix this we need to change the execution order of our page and change the function of our script to POST actions rather than GET actions.

### 8.1. Create the new file and move your functions

In order to create our new file, we simply copy the complete functions from our main PHP page and place them into a new text file. We need to move the following functions:

- Global variables
- translateXML
- feedOptions
- navigation
- showFeed
- subscribe
- unsubscribe

Move these functions from your main PHP page into a new file.

### 8.2. Create an init function in the new file

With our functions in a new file, we need to change our code in order to accomplish three things. First, we need to change our page flow (when to call the subscribe unsubscribe, and showFeed functions. Second, we need to move some of this page flow to an initialize function. Create a new function called init() in the PHP functions file. Use the select case statement from our main page as the basis:

```
function init() {
    global $server, $dbserver, $username, $pass, $database;
    switch ($_REQUEST['pageProcess']) {
        case "Subscribe":
            subscribe();
        break;
        case "Unsubscribe":
            unsubscribe();
        break;
    }
}
```

### 8.3. Calling our new file

Place a call to this function in the top line of your main PHP page. This function needs to be before even the doctype declaration so that we can set document headers in later examples

```

<?php
    require('./ex8_src.php');
    init();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

```

#### 8.4. Rewrite the PHP code in the main div

Now that our functions and page logic are located in an external file, we need to place a call to showFeed in our main div in our PHP page. Modify your main div to look like the following code.

```

<div id="main">
    <?php if($_REQUEST['pageProcess'] == 'MYSQL_showFeed') showFeed();?>
</div>

```

#### Contents of the new PHP\_functions.php file

```

<?php
$server = "http://www.ils.unc.edu".$_SERVER['PHP_SELF'];
$dbserver = "pearl.ils.unc.edu";
$username = "inls572_spring08";
$password = "yreuq572";
$dbname = "inls572_spring08";
/*This is our new function init(). It will allow us to call our PHP functions only when necessary. We will call this init
function in the header of our main PHP page from now on */
function init() {
    global $server, $dbserver, $username, $password, $dbname;
    switch ($_REQUEST['pageProcess']) {
        case "Subscribe":
            subscribe();
        break;
        case "Unsubscribe":
            unsubscribe();
        break;
    }
}

function translateXML ($xmlinput, $xslfile, $params) {
    $xmlfile = domxml_open_file($xmlinput);
    $xslt = domxml_xslt_stylesheet_file($xslfile);
    $output = $xslt->process($xmlfile, $params);
    echo $output->dump_mem();
}

/*Our new function feedOptions()*/
function feedOptions() {
    global $server, $dbserver, $username, $password, $dbname;
    $connection = mysql_connect($dbserver, $username, $password);
    mysql_select_db($dbname, $connection);
    //Create a query to get data from the database, store the output in the $result variable
    $result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);
    //Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
    while ($row = mysql_fetch_array($result, MYSQL_NUM))
    {
        //Write a mix of HTML and database content to the output variable. In this case it is an option element for our form
        $output .= '<option value="'. $row[0]."'>'. $row[2]. '</option>';
    }
}

```

```

//close your output variable
//echo your output variable to the browser
echo $output;
}

/* We have modified our navigation function just a bit to use the global variables. Otherwise, nothing changed here */
function navigation () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);

    //Create a query to get data from the database, store the output in the $result variable
    $result = mysql_query("SELECT * from feeds where username = 'mitcheet'", $connection);
    //Begin storing html in an variable that we can print out later - for now just open our Navigation UL
    $output = '<ul class="navigation">Subscribed Feeds';
    //Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
    while ($row = mysql_fetch_array($result, MYSQL_NUM))
    {
        //Write a mix of HTML and database content to the output variable
        $output .= '<li><a href="'. $server. '?pageProcess=MYSQL_showFeed&feed='. $row[0]. "'>'. $row[2]. '</a></li>';
    }
    //close your output variable
    $output .= '</ul>';
    //echo your output variable to the browser
    echo $output;
}

/* Same thing for showFeed. Only our use of global variables changed here */
function showFeed () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
    //Create a query to get data from the database, store the output in the $result variable
    $result = mysql_query("SELECT * from feeds where id = ".$_REQUEST['feed'], $connection);
    //Use a while statement to get data from the MYSQL database. This will return each row from your query one at a time
    while ($row = mysql_fetch_array($result, MYSQL_NUM))
    {
        $params = array('feedUrl' => $row[3], 'pageProcess' => $_REQUEST['pageProcess']);
        translateXML ($row[3], './templates.xml', $params);
    }
}

/* Our two new functions, subscribe and unsubscribe use the insert and delete functions of SQL but otherwise, look very similar to
our other two MySQL functions */
function subscribe () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
    /* Notice the syntax of our Insert SQL statement - Insert into table (column list) values (value list); */
    $SQL = "INSERT INTO feeds (username, feedname, feedURL) values ('".$_REQUEST['username']. "',
'".$_REQUEST['feedName']. "', '".$_REQUEST['feedUrl']. "')";
    $result = mysql_query($SQL, $connection);
    /*Lets delete the echo result function below */

    echo $result;
}

/* Here is our unsubscribe function. Notice the syntax of the delete statement delete from table where criteria = ""*/
function unsubscribe () {
    global $server, $dbserver, $username, $pass, $database;

```

```

$connection = mysql_connect($dbserver, $username, $pass);
mysql_select_db($database, $connection);
    $SQL = "DELETE from feeds where id = ".$_REQUEST['feedId'];
    $result = mysql_query($SQL, $connection);
/*Lets delete the echo result function below */
    echo $result;
}
?>

```

**Our new PHP file follows. Notice the two bolded comments for changes from the previous file.**

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
<!-- Notice the new inclusion of our PHP require and init command -->
    <?php
      require('./php_functions_ex7.php');
      init();
    ?>
    <title>RSS Feed Aggregator </title>
    <link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
    <meta name="DC.title" content="RSS Feed reader"/>
    <meta name="DC.author" content=" "/>
    <link rel="stylesheet" type="text/css" href="/main.css"/>
  </head>
  <body>
    <div id="header">
      <h1>RSS aggregation service</h1>
    </div>
    <div id="nav">
      <?php navigation(); ?>
<!-- For the time being, lets hard-code our tasks in as members of an unordered list Notice our use of the hidden form element to limit which username is used to subscribe/unsubscribe -->
      <ul>Tasks
        <li>Subscribe to a feed
          <div id="subscriptionform">
<!--Notice how on the form action, we reference our global variable -->
            <form action="<?php global $server; echo $server;?>" method="post">
              <fieldset>
                <label for="feedURL">Feed URL</label><input type="text" name="feedUrl"/>
                <label for="feedName">Feed Name</label><input type="text" name="feedName"/>
<!--Put your own username in on the next line -->
                <input type="hidden" name="username" value="mitcheet"/>
                <input type="submit" name="pageProcess" value="Subscribe" />
              </fieldset>
            </form></div>
          </li>
          <li>Unsubscribe from a feed
            <div id="unsubscriptionform">
              <form action="<?php global $server; echo $server;?>" method="post">
                <fieldset>
                  <input type="hidden" name="username" value="mitcheet"/>
                  <select name="feedId">
<!-- We will use our MySQL functions to help create a list of feeds to unsubscribe from Call the function here, we will define it below-->
                    <?php feedOptions(); ?>
                  </select>
                  <input type="submit" name="pageProcess" value="Unsubscribe" />
                </fieldset>
              </form>
            </div>
          </li>
        </ul>

```

```
        </fieldset>
    </form>
</div>
</li>
</ul>
</div>
<div id="main">
<!-- In our Main processing area, are just checking to see if we need to show a feed. -->
    <?php if($_REQUEST['pageProcess'] == 'MYSQL_showFeed') showFeed();?>

</div>
</body>
</html>
```

## Exercise 9. Use JavaScript to control form display

One way of keeping interfaces simple is to use JavaScript to show page content/functions only when necessary. By using JavaScript to hide and place forms we can keep our interface clean while adding functionality. While we could accomplish this task using CSS, JavaScript provides us more methods to work with.

In order to accomplish this, we will need to create a JavaScript function that will modify the display style and position of our form elements. We will access our form elements by referencing the div ids (subscription and unsubscription). By approaching JavaScript in this manner (using JavaScript to detect HTML elements and place behaviors on them) our page will function both with and without JavaScript.

### 9.1. Create the showHide function

We will begin by inserting a `<script language="JavaScript">` into our head tag. Then create the function showHide which accepts four parameters: element (the name of the element to modify), visibility (either visible or hidden), position (relative or absolute), and left (the amount of offset from the left hand side of the document). You should recognize these elements from our work with CSS earlier in the semester. Enter the following code in the script tag.

```
<script language="javascript">
/*This function takes four parameters - elementid visibility, position, and left offset. They use the getElementById function
to browse the DOM to these elements and then set their properties. The element ID are assigned to a wrapping div on the
form elements*/
function showHide(elementid, visibility, position, left) {
    form = document.getElementById(elementid);
    form.style.visibility = visibility;
    form.style.position = position;
    form.style.left = left;
}
</script>
```

### 9.2. Modify HTML

Now that we have created our JavaScript code, we need to add a few links to the form to allow users to show and hide the forms. We will do this by hard-coding javascript function calls into the links. While not absolutely following the DOM approach, because our forms will work without JavaScript this is not a major issue. See the HTML code in our nav div below and make the changes indicated in Bold. Notice the addition of a ‘Subscribe to a feed’ link and a cancel link. The two links are implemented in different ways but both are valid JavaScript. Notice that it is important for you to return ‘false’ in the second example.

```
<div id="nav">
  <?php navigation(); ?>
  <!-- For the time being, lets hard-code our tasks in as members of an unordered list Notice our use of the hidden form element to
  limit which username is used to subscribe/unsubscribe -->
  <ul>Tasks
    <li><a href="javascript:showHide('subscriptionform', 'visible', 'relative', '0px')">Subscribe to a feed </a>
      <div id="subscriptionform">

  <!--Notice how on the form action, we reference our global variable -->
    <form action="<?php global $server; echo $server;?>" method="post">
      <fieldset>
        <label for="feedURL">Feed URL</label><input type="text" name="feedUrl"/>
```

```

        <label for="feedName">Feed Name</label><input type="text" name="feedName"/>
<!--Put your own username in on the next line -->
        <input type="hidden" name="username" value="mitcheet"/>
        <input type="submit" name="pageProcess" value="Subscribe" />
        <a href="#" onclick="showHide('subscriptionform', 'hidden', 'absolute', '-500px'); return
false;">Cancel</a>
        </fieldset>
    </form></div>
</li>
<li><a href="javascript:showHide('unsubscriptionform', 'visible', 'relative', '0px')">Unsubscribe from a feed </a>
    <div id="unsubscriptionform">

        <form action="<?php global $server; echo $server;?>" method="post">
            <fieldset>
                <input type="hidden" name="username" value="mitcheet"/>
                <select name="feedId">
<!-- We will use our MySQL functions to help create a list of feeds to unsubscribe from Call the function here, we will define it
below-->
                    <?php feedOptions(); ?>
                </select>
                <input type="submit" name="pageProcess" value="Unsubscribe" />
                <a href="#" onclick="showHide('unsubscriptionform', 'hidden', 'absolute', '-500px'); return
false;">Cancel</a>
            </fieldset>
        </form>
    </div>
</li>
</ul>
</div>

```

### 9.3. Create a JavaScript initialization function

Now that we have a function that will show and hide our forms, we need to create an initialization function to hide our forms by default. We will accomplish this by defining a simple function called `init()` which contains two calls to our `showHide` function. We will then use the `onload` method of the window object to call the `init` function. The `window.onload` method is used specifically to call functions that you want to run when the webpage has finished loading. Enter the following code into the `<script>` element of our PHP file.

```

function init() {
    showHide('subscriptionform', 'hidden', 'absolute', '-500px');
    showHide('unsubscriptionform', 'hidden', 'absolute', '-500px');
}

window.onload = init;

```

### 9.4. Add some style to our form links

In our CSS file, let's style our anchor references in the form a bit

### 9.5. Moving our JavaScript into a new file

Now that we have some JavaScript code, we need to create an include file to hold it. Create a new text file and copy the contents of the script tag to a new text file (do not include the script tags). Our new file should contain the following code:

```
function showHide(elementid, visibility, position, left) {
    form = document.getElementById(elementid);
    form.style.visibility = visibility;
    form.style.position = position;
    form.style.left = left;
}

function init() {
    showHide('subscriptionform', 'hidden', 'absolute', '-500px');
    showHide('unsubscribeform', 'hidden', 'absolute', '-500px');
}
window.onload = init;
```

Update the script tag in your PHP page to:

```
<script src="./js_functions_ex9.js"/>
```

## Exercise 10. Adding a Login function to our page

Now that we have the ability to subscribe to, view, and unsubscribe from feeds, we need the ability to support multiple users. Up to this point, we have hard-coded our username into forms so that we can all play in the same database. In this exercise, we will create login and logout functionality and use cookies to remember our username. In addition, our page will contain logic which will prevent it from showing or modifying data without a login. The authentication method used in this example is very loose (basically no authentication). In more complex environments, you would also ask for a password and complete a database lookup to verify the user.

### 10.1. Create a login form

First, let's create a form in our header div that has a single text field (username). Below is our entire header div:

```
<div id="header">
  <h1>RSS aggregation service</h1>
  <!-- Let's use a simple form and loose authentication to make our page usable by multiple people. -->
  <form action="<?php global $server; echo $server;?>" method="post">
    <fieldset>
      <label for="username">Username</label><input type="text" name="username"/>
      <input type="submit" name="pageProcess" value="Login" />
      <input type="submit" name="pageProcess" value="Logout" />
    </fieldset>
  </form>
</div>
</div>
```

Now, let's add some style to our main.css file to style our form a bit:

```
#loginform { float:right; display:inline; }
```

### 10.2. Create login and logout functions

Now we need to create the PHP function that will take the login and logout information. In a true authentication environment, you would take a username and password and look that combination up in a database or external authentication system. In our case, we will just use the username as an identifying mechanism and use client side cookies to remember the login.

Create two new functions in your PHP include file (login and logout). Both of these functions set a cookie with the value from your login application and redirect the page to itself in order to make sure the page loads knowing that the user is logged in our out.

```
function login () {
  /*In this function we will just write a cookie to the browser with your login infomration. In a more complex example you would verify the login and perhaps retrieve user settings from a database. The setcookie function in PHP takes five variables. In our case, we will use the first three - cookie name (inls572_username), value (the value from our login form), and the expiration time (the current time plus 3600 seconds */
  global $server;
  $result = setcookie('inls572_username', $_REQUEST['username'], time()+3600);
  header("Location: ".$server);
}
```

```
function logout() {
    global $server;
    $result = setcookie('inls572_username',"", time()-3600);
    header("Location: ".$server);
}
```

### 10.3. Add login/logout logic to your page

Now we need to modify our script to process the login/logout functions. We will accomplish this by checking two things. First, we will check to see if the pageProcess being called is either Login or Logout. If so, we will call those functions. By default, these functions will complete the function and refresh the page. The other thing we will do is check to see if we have a cookie value for inls572\_username. If we do, we will use that value to control the show, navigation, subscribe, and unsubscribe functions. You can see in the init() function below, we first check for a cookie, then check to see if the page is trying to login. This will stop the page from performing operations if there is no username set in the cookie. Make the following modifications in the PHP functions page you created in exercise 8.

```
function init() {
    global $server, $dbserver, $username, $pass, $database;
    /*in this If statement, we check to see if we need to process the login or logout functions */
    if ($_REQUEST['pageProcess'] == 'Login') {
        login();
    } elseif ($_REQUEST['pageProcess'] == 'Logout') {
        logout();
    }

    /*We use the isset function to check if we have a cookie called inls572_username*/
    if (isset($_REQUEST['inls572_username'])) {
        switch ($_REQUEST['pageProcess']) {
            case "Subscribe":
                subscribe();
                break;
            case "Unsubscribe":
                unsubscribe();
                break;
        }
    } else {
/*if we do not have the cookie, call a function to print the login message */
        loginmessage();
    }
}
```

### 10.4. Create the loginmessage function

Now we need to create the login message function to tell users to login. Create this in your functions page as well.

```
function loginmessage() {
    echo '<div class="login">Please login to use this page</div>';
}
```

## 10.5. Update PHP functions to reference the login username

Now that we have login/logout functionality, we need to tell our PHP functions about it. We need to modify elements of our feedOptions, navigation, and subscribe functions to include references to our new username element.

```
function feedOptions() {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
    //Create a query to get data from the database, store the output in the $result variable
    $result = mysql_query("SELECT * from feeds where username = '".$_REQUEST['inls572_username']."'",$connection);
    ....the rest of the feedOptions function

function navigation () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
    //Create a query to get data from the database, store the output in the $result variable
    $result = mysql_query("SELECT * from feeds where username = '".$_REQUEST['inls572_username']."'",$connection);
    .....the rest of the navigation function

function subscribe () {
    global $server, $dbserver, $username, $pass, $database;
    $connection = mysql_connect($dbserver, $username, $pass);
    mysql_select_db($database, $connection);
    /* Notice the syntax of our Insert SQL statement - Insert into table (column list) values (value list); */
    $SQL = "INSERT INTO feeds (username, feedname, feedURL) values ('".$_REQUEST['inls572_username'].',
''".$_REQUEST['feedName'].', ''".$_REQUEST['feedUrl'].')";
    ....The rest of the subscribe function
```

## 10.6. Show who is logged in

Finally, we need to modify our Login form to allow for logout functionality and to show the user who is logged into the page. In the header, inside the loginform div, add a PHP call that gets the username and prints out a message telling us who is logged in:

```
<div id="loginform">
    <?php if( isset($_REQUEST['inls572_username'])) echo '<p>You are logged in as
    '".$_REQUEST['inls572_username'].'</p>';?>
```

Finally, lets style up our login message in our css file:

```
.login {
    text-align:center;
    height: 30px;
    background-color:#CCFFFF;
    font-weight: bold;
}
```

## Exercise 11. Introducing Ajax

In this exercise, we will begin working with Ajax to take over part of the functionality of our page. Because Ajax is an advanced interaction technique, it should always be built as an addition to basic page functionality. This exercise has several components and requires building a number of JavaScript functions. The final result is the ability to use Ajax to request feed updates from the server without a full page refresh. The outline of this exercise is:

1. Create the Ajax XMLHttpRequestObject
2. Create a function that requests and returns data via the XMLHttpRequestObject
3. Create functions that will shuttle requests and data through the first two functions in order to complete page tasks.
4. Add interactivity functions to make the interaction obvious to the user.

Throughout these exercises, we will be adding code to our JavaScript include file.

### 11.1. The Ajax XMLHttpRequestObject

In this part of the exercise we will create a new function called `getHTTPObject()`. This function essentially completes a browser detection function by checking to see if certain methods (the XMLHttpRequest (mozilla) and ActiveXObject (Microsoft)) are supported by the window object. The variable `xhr` is an object that is created by this function and returned at the end. This function will be called every time we complete an Ajax request.

```

/*Our Base Ajax Function. This does a simple browser detect */
function getHTTPObject() {
    var xhr = false;
    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }
    return xhr;
}

```

### 11.2. The Ajax data interaction function

In our example, this function handles the process of sending requests to our PHP scripts for transformed RSS feeds. We will pass this function two parameters: `file` (the name of the script to call), and `query` (the querystring to pass it).

```

function updateFeed(file, query) {
/*Call our Ajax initialization function */
    var request = getHTTPObject();
/*if our object is returned successfully then start a request */
    if (request) {
/*Create a generic function to attach to the change method of our Ajax request object. This function will watch the Ajax process and process content once it is retrieved */
        request.onreadystatechange = function() {
            displayResponse(request);
        };
    }
}

```

```

/*Begin the request, pass it the processing file using the POST method and perform the task asynchronously (the value true */
    request.open("POST", file, true);
/*For Post request, we have to set a content header */
    request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
/*Send the request */
    request.send(query);
/*Provide a return value for this function so that it can be accessed later. True means that the function was able to start the process,
false means that no XMLHttpRequestObject could be created */
    return true;
    } else {
    return false;
    }
}

```

### 11.3. The Ajax displayResponse function

The displayResponse function watches the Ajax request and waits for it to get all of its data back. It then modifies the contents of our HTML document to display the data it retrieved.

```

/*We pass this function our Ajax request object */
function displayResponse(request) {
/*We check the state of our connection, 0=uninitialized, 1=loading, 2=loaded, 3=interactive, 4=complete*/
    //if the connection is complete
    if (request.readyState == 4) {
    //if the connection was successful - notice the use of http response codes
        if(request.status == 200 || request.status == 304) {
/*Define an object called details and set it to the value of the main ID in our DOM */
            var details = document.getElementById("main");
/*Cheap & Quick, set the contents of our div to the data returned by our PHP script. This works in our case because the PHP script
is returning an unsorted list of RSS content. In more complex cases you may have to do client side processing on this data */
            details.innerHTML = request.responseText;
/*The fadeUp function is used to show the user that something happened. We will get to that function in a minute */
            fadeUp(details, 255,255,153);
        } else {
/* if something went wrong, tell us */
            alert(request.status);
        }
    }
}
}

```

### 11.4. Indicating Interactivity

There are a number of ways to show the user that something happened. Two popular methods are animated graphics while data is being retrieved and color changes on new data. The following function uses the color change approach by highlighting changed data with a color change which fades out over the course of a few seconds. This is not specifically an Ajax function but it does tend to be used in Ajax environments. This is an advanced function which uses recursion some window methods. If this one seems too confusing, that is ok, just type it in or skip it for now and comment out the call to fadeUp in the other functions.

```

/*We pass this function an element object reference, and three values for RGB levels) */
function fadeUp(element, red, green, blue) {
    if (element.fade) {
/*Clear the timeout on the element */
        clearTimeout(element.fade);

```

```

    }
    /*Set the style on the element background */
    element.style.backgroundColor = "rgb("+red+", "+green+", "+blue+)";
    if (red == 255 && green ==255 && blue == 255) {
        return;
    }
    /*change the values of our red, green, and blue variables */
    var newred = red + Math.ceil((255-red)/10);
    var newgreen = green + Math.ceil((255-green)/10);
    var newblue = blue + Math.ceil((255-blue)/10);
    /*Use recursion to call this function again until all values of red, green, and blue are equal to 255 */
    var repeat = function() {
        fadeUp(element, newred, newgreen, newblue)
    };
    /*Define the element.fade method */
    element.fade = setTimeout(repeat, 100);
}

```

## 11.5. Our new initialization function

One of the basic premises of Ajax is that it should enhance and not replace server-side functionality (when possible). For this reason, rather than hard-coding our JavaScript Ajax calls on to our links, we will use a JavaScript initialization function to add the Ajax call to our RSS feed links. By approaching the integration in this way, our program will work just fine without client side JavaScript:

```

function init() {
    /*Remember our showHide function from before. Be sure to include them here so our forms don't show */
    showHide('subscriptionform', 'hidden', 'absolute', '-500px');
    showHide('unsubscribeform', 'hidden', 'absolute', '-500px');

    /*Define an object nav which refers to our navigation div */
    var nav = document.getElementById("nav");
    /*Access the UL element within our nav object */
    var myul = nav.getElementsByTagName("ul")[0];
    /*Access the anchor references in our UL object */
    var navLinks = myul.getElementsByTagName("a");
    /*For each anchor element in our object ...*/
    for (var i=0; i<navLinks.length; i++) {
        /*Create a generic function to attach to the onclick event */
        navLinks[i].onclick = function() {
            /*This function will grab the href of the current anchor reference and split it apart */
            var query = this.getAttribute("href").split("?")[1];
            /*We will isolate the feed id portion of the querystring */
            var feedid = query.split("&")[1];
            /*We will rewrite our URL to use the Ajax page process */
            var url = "pageProcess=AJAX_showFeed&"+feedid;
            /*Finally we will return the opposite value of our updateFeed function. Because our updateFeed function returns true or false based
            on the success or failure of our Ajax Object, This has the net effect of not updating our links when the call to updatefeed Fails. */
            return !updateFeed('./php_functions_ex10.php', url);
        }
    }
}

```

## 11.6. Modify our PHP\_function file to recognize the ajax call:

Finally, we need to let our PHP function file know that it might be getting Ajax calls. We will do this by telling it about what the querystring from an Ajax call looks like (using the `pageProcess` variable). Insert the following lines into your PHP function source **outside of any function call**. You should insert them just below your global variable definition:

```
/*Let Ajax Request take control */
switch($_REQUEST['pageProcess']) {
    case "AJAX_showFeed":
        showFeed();
        break;
}
```

We also need to add some processing to our XSL template (remember that file?). Due to the way we wrote our choose statement in our XSL templates we need to add a case for when the `pageProcess` is equal to `AJAX_showFeed`. Add the following lines to our xsl template file:

```
<xsl:when test="$pageProcess = 'AJAX_showFeed'">
    <xsl:call-template name="MYSQL_showFeed"/>
</xsl:when>
```