First Year Experience
Teaching Assistant
Evaluation Website

https://fyp.nmu.edu/

For

Network Computing

Senior Project

By Jeff Wolf

From

April 7, 2006 - December 7, 2006

Introduction

First off I would like to talk a little bit about myself. I am a Senior at NMU majoring in Network Computing with a minor in Mathematics. I chose the Network Computing degree because I like the concept of networking and I liked the idea of being able to take Electronic imaging courses towards my degree. I took courses such as Web Application Programming, Website Design, Unix System Administration and Network Computing because I am interested in everything about the Internet. My favorite of these courses was Web Programming where I learned the server side scripting language PHP and MySQL database. PHP became my favorite language because it blends seamlessly with HTML and MySQL. Combined, PHP and MySQL can create many interesting dynamic websites and web applications which is why it is used in many of the sites and applications I use everyday. Examples of PHP web applications are phpBB, Gallery, and phpMyAdmin to name a few.

I have worked at the Dean of Students Office since May of 2005 as their webmaster/developer/systems administrator/technical help desk, well basically I am their "tech guy" but my main focus is managing the DSO website and web server. At the DSO I was thrown into managing a large dynamic PHP and MySQL website and Unix server. I leaned quickly what it meant to manage a website and saw just what PHP could do first hand. My skills improved vastly as I was forced to create different functions and applications requested by DSO staff for the website. But I still wasn't required to create a large project from scratch. I was lucky enough to work off of previous code which helped a great deal towards developing my style.

Project Overview

My Senior Project idea came to me when a co-worker of mine at the Dean of Students Office asked if it would be possible to develop an on-line form for First Year Experience Students to evaluate their Teaching Assistants. After a while of us brainstorming ideas I was convinced that this would work as a good Senior Project. Since my main focus in my Network Computing degree has been web development including programming, databases, networking, system administration and even web design, I though it would be a great opportunity to showcase and develop my skills in one large

project.

The original idea was to have a dynamic form for the students and an administration site for the FYE Coordinator and Graduate Assistant to manage the form and results. It quickly grew into the website I have developed, with multiple user roles, more user control over the site, and a complete database of students, teachers, assistants, blocks, rooms, and more. Threw out the project, my main goal was to create a site the can easily be managed by people who know little about HTML or PHP. With the exception of adding new features (which i made quite easy by compartmentalizing the code into files and files into folders) the coordinator never needs to look at any code or write any database queries to manage most aspects of the site.

Project Elements

Hosting and Security

Hosting is a small but critical part of this project. I host the website on the DSO's FreeBSD 5.3 web server where I also manage *dso.nmu.edu*, *asnmu.nmu.edu*, *eso.nmu.edu* and now *fyp.nmu.edu*. The server runs Apache 2.0 with SSL and MySQL 4.0.21 along with PHP 5.0.2 and Perl. I primarily used PHP and MySQL for this Project with some JavaScript as needed. I created the virtual CNAME off of *dso.nmu.edu* for the alias *fyp.nmu.edu* for the website to keep it separated and make it easy to find for users. Basically all this means is that there is a folder called *fyp* on the server that is mapped to the address fyp.nmu.edu so when people go there they see the contents of this folder, rendered by the browser.

By adding an "s" after "http" on all of the addresses it tells the browser to use SSL (Secure Socket Layer). SSL makes sure any of the information passes between the user's computer and the server is completely secure because no one can snoop or intercept any unencrypted information. Basically this is possible by creating something called an SSL certificate which proves to the user that "I am who I say I am" and that their information will be encrypted. Once I created the sub-domain for fyp I needed to make a new SSL certificate to match the new name or else the browser would throw a mismatching name certificate error. This is something I tried but was unable to complete. I created the new certificate with the name of fyp.nmu.edu and added it to the apache configuration files.

However the website is still displaying the old certificate with the name of dso.nmu.edu. So the browser throws a warning out before the user can accept the certificate. However this does not stop any of the security because after the user accepts the certificate it is completely secure. I learned a lot about SSL for this project like creating and signing certificates for example. I also learned about creating virtual aliases with apache configuration files.

Database

I knew that the database would be a significant factor in the success of this site so I began work on the structure first. After one major redesign a few months into the project I had the final design (*See Illustration 1*). Much of the structure revolves around the *Semesters* table. This is because the site will be used to span many semesters to come and instead of overwriting data each semester I wanted to be able to store all the old evaluation results for future comparisons. The only tables that are not linked to the *Semester* table are *Todo*, *Users*, *Navbar*, and *Navbar_Headers* because they stay unchanged when a new semester is created.

This is where the paper gets a bit technical and boring but please bare with me as I try and explain the database structure and why I did it this way. It may help to look at *Illustration 1* (page 6) while reading this section. The Evaluation Form and results are handled by a total of three tables, *Questions*, *Question_Answers*, and *Question_Headings*. Each question has its own heading and semester. Each question_answer has its own question and each heading has its own semester. Finally, each question_answer is linked to a TA in the *Teaching_Assistants* Table but nowhere is it linked to the students because the student evaluation must be anonymous. This structure allows the form to be different for each semester if need be and questions to be grouped into sections.

The other major table that ties the database together is the *Blocks* table. Since FYE courses are organized into blocks and each student is place in a block of classes, I used the same concept for my database. The tables that are linked to the *Blocks* table include, *Teaching_Assistants*, *Instructors*, *Students*, *UN100*, and *Semesters*. The *Students* table has a *block_id* column because a student will never be in more then one block. However it is possible for a TA or Instructor to be assigned to more then one block therefor the *Blocks* table contains the *ta_id* and i*nstructor_id* columns. Since each block also has one semester it is possible to tell which semester a student is in by knowing what block they

are in. It is also worth noting that the *Students* table contains an *evaluated* column so it can keep tract of weather a student has already submitted an evaluation. It's primary key is the *nmuin* which is how the student loges in to submit their form.

To digress a bit, I will explain a problem I encountered with this design... By having the primary key of the students table be the NMU Identification Number or nmuin it requires that the database have every student's nmuin. To fill the student table for this semester I got an excel sheet of all the students enrolled in the FYE Program and wrote a script to import them into the database. Unfortunately the excel sheet did not include the nmuin's and the only way to get them is to use Banner which I don't have access to. Most people in the office do have access but were unable to export the students and their nmuin's in a meaningful way for me to import them into my database. So I had the FYE Coordinator request an Impromptu report with the data I need. I am still waiting on this report.

The reason I chose to have the students table is that I needed a way to keep the evaluation fair. I could not let just any student submit their evaluation for any TA they want and also I couldn't have them submitting more then one evaluation. It would make the whole project irrelevant if the data could not be trusted. So my first solution was to use the TA's table. When the TA would be added to the database by the FYE Staff, a password would be automatically generated and emailed to the TA along with instruction on what to do with it. It would say that on the last day of classes they could tell their students the user name and password given to them. The students would then log in and could submit their evaluation in class. I had this system up and working but it wasn't perfect. Yes it would stop student from evaluating other TA's and it was completely anonymous for the students, but it didn't stop the from submitting multiple evaluation and the major problem. The major problem was that it left the reliability in the trust of the TA. With the TA having the user name and password they could submit as many evaluations for themselves as they want. This was not acceptable so I came up with the solution of having a database of students and allowing the student to type in their nmuin to submit the evaluation. This was perfect, it solved all the problems. The student's nmuin is supposed to be kept secret, I can make sure each student only enters one evaluation for only one TA and only the TA's Students can evaluate them. Also since I don't link the student to their evaluation results it is anonymous. The only caveat is that I need access to the students' nmuin's. In the next

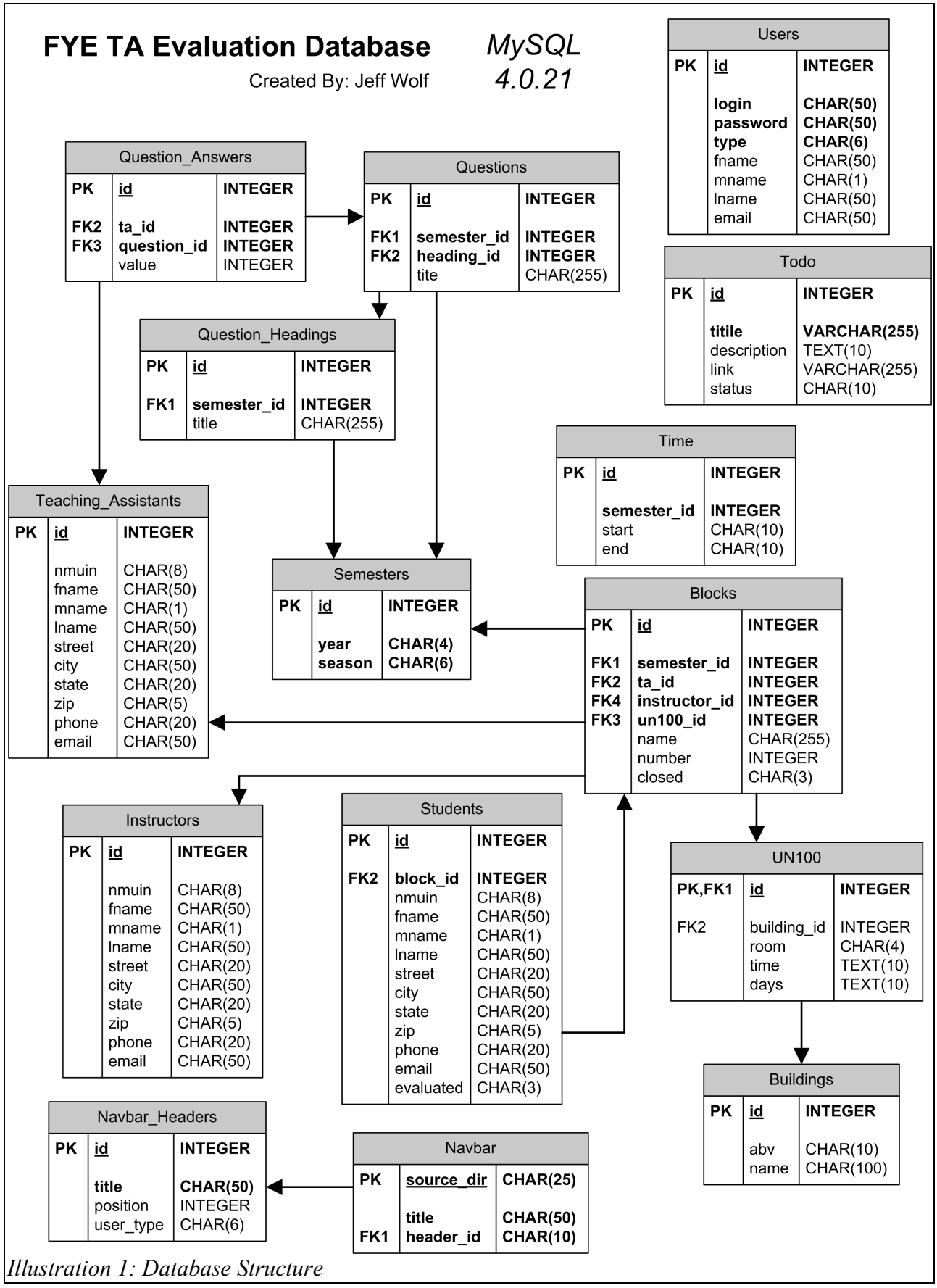

Illustration 1: Database Structure

semester when a student is accepted to the FYE the staff will input them into the database as they currently do but this time it will be the new database that I created.

I added the *Navbar* and *Navbar_Headers* tables late in the project because I got tired of changing the navigation bar HTML code each time I wanted to add a new link. So now the navigation bar is controlled threw the web interface. The *Users* table has a *type* column so that I can have multiple user roles with different access to parts of the site. I also recently added a *Time* table so that the the evaluation form can only be accessed at a certain time period as set by the coordinator.

In the database portion of this project I learned a lot from my mistakes. My original database had a few poor design ideas which cost me time later on. When I sat down and really thought out the structure with Visio is when I think I came up with a good design.

File Structure

On any large scale website I do, it is important for me to have a well organized file structure. It allows for faster coding, and makes it easier to update later on. I like to keep all of the website elements (CSS files, fonts, images...) in a separate location so they don't get confused with the actual web pages. All of the web pages are separated first by the name of the user role that can access those files. For example, the FYE Coordinator can access everything in the *admin* and *grad* folders, the Graduate Assistant can only access the *grad* folder, and the Web Master can access *admin*, *grad* and *webmaster* folders. The only user that can get into the *Students* folder and therefore submit an evaluation is an FYE Student. The name of these folders are

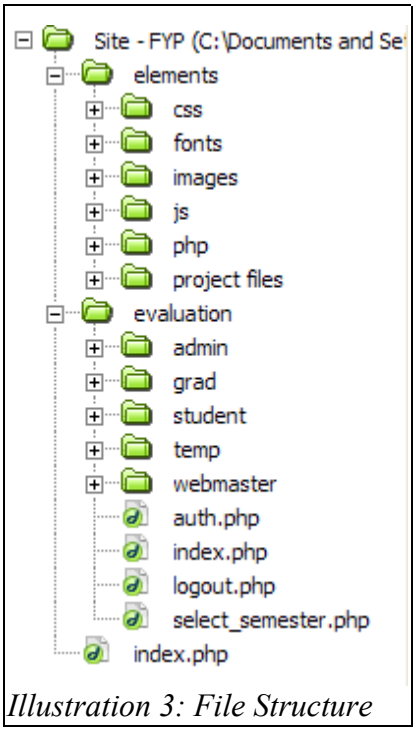

Illustration 3: File Structure

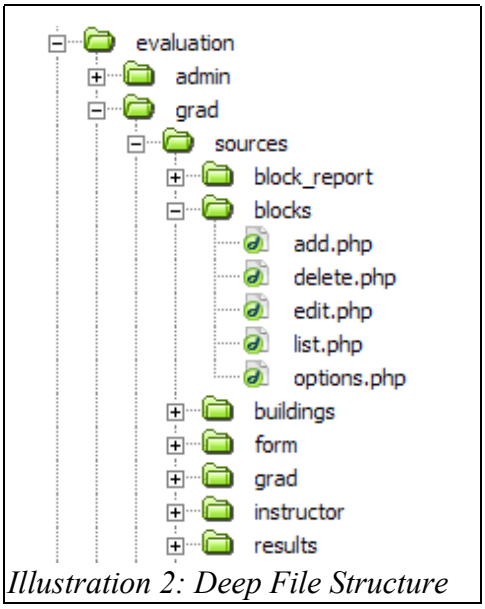

Illustration 2: Deep File Structure

special and must also be changed in the *Navbar_Headers* table. However it is easy to modify and add new users and roles.

Within each user folder there are two files (*index.php* and *load_session.php*) and a folder called *sources*. Within the *source* folder is a *source_handler.php* file and a folder for each link (or source) in the navigation bar. And within each link folder are the files that control that link's functions. This way the sites structure represents the file structure itself for convenience. This makes it easy to add new links just by copying a source folder, renaming it and changing the files within to represent that links new functionality.

```php
<?php
  //load session for Coordinator
  session_start();

  if(isset($_SESSION['type']))
    {
    if($_SESSION['type']=="Admin" or $_SESSION['type']=="Master")
      {
      if(isset($_SESSION['semester']))
        {
        require_once("var/www/fyp/elements/php/classes/class.dbXML.php");
        $db = new dbXML("Evaluation","dso","d5o04n05");
        }
      else
        print "<META HTTP-EQUIV=\"refresh\" CONTENT=\"0; url=https://fyp.nmu.edu/evaluation/select_semester.php\">";
      }
    else if($_SESSION['type']=="Grad")
      print "<META HTTP-EQUIV=\"refresh\" CONTENT=\"0; url=https://fyp.nmu.edu/evaluation/grad/index.php?referer=Grad\">";
    else if($_SESSION['type']=="TA")
```

Code View 1: Load Session

With this system, when a user goes to *fyp.nmu.edu/admin* for example, the first file to load is the *index.php* from the admin directory. The first line of *index.php,* includes the load_session file. As you can see in *Code View 1*, the load_session file starts the session and makes sure the user has the right permissions. If he/she does, the main database class is loaded and the $db variable is initialized for easy use later. If the user is of a different role, they are redirected to their user's folder. Once successful, the rest of the *index.php* can be displayed (see *Code*

```php
    <div id="header">
    <? include("var/www/fyp/elements/php/header.php"); ?>
    </div>
    <div id="header2"></div>
    <div id="options">
    <?php
      $source = $_REQUEST['source'];
      if($source == "") print '<strong>Welcome '.$_SESSION
      else include("sources/$source/options.php");
    ?>
    </div>
    <div id="wrapper">
      <? include("var/www/fyp/elements/php/navbar.php"); ?
      <div id="content">
        <? include("sources/handler.php"); ?>
      </div>
    </div>
  </body>
```

Code View 2: index.php

View 2).

The header is the top graphic in the website that contains the title, login and log out controls and the semester select control. There is one file called *header.php* in the /elements/php/ directory that controls all that is displayed in the header. The Options bar as I call it is the small protrusion below the header that contain options specific to each source or link (*see Illustration 4*). Which options are

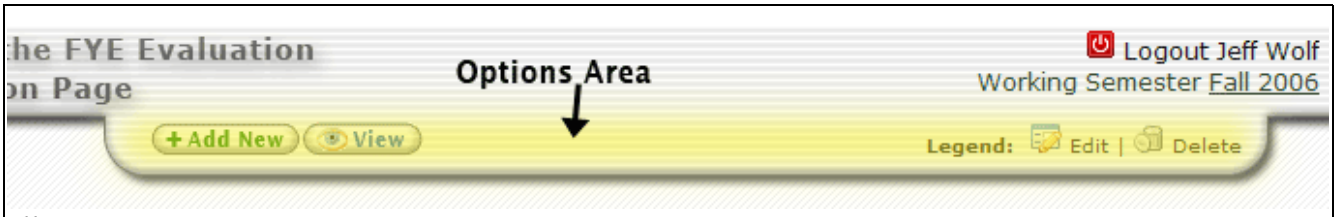

Illustration 4: Options Bar

displayed is determined by which source was requested so the *options.php* file is loaded from within the correct link's folder. The Navigation bar is loaded next which I will talk about later on in this paper. Finally the *source_handler.php* file is called to control which source or link is to be displayed.

This all happens to build one HTML file that is displayed to the user. Since the files are compartmentalized only the code that is needed will be loaded thus saving on server cost and bandwidth for the user. It also make for easy maintainability because locating a problem file is easy. Plus it is easy to add features by duplicating a similar function. I realized that there was too much repeating code with this method. So I started finding similar code that repeats and creating functions out of it. The result of this is the navbar, header, and the *get_edit_sting* and *get_add_string* functions which I will discuss in the XML section. This cleaning up of the code made it easier to read and understand.

PHP and XML

In previous web projects I have used a PHP database class that creates and preforms all the queries. I usually pass the class associative arrays which are parsed by the class to create update, and insert queries and the select queries return associative arrays. That system works fine but for this project I wanted to do something different. At the same time I was seeing a lot of RSS and XML used on the web so I thought I better learn how to use it myself. This is how I came up with my idea to pass the db class XML strings instead of associative arrays.

The main class in my site is *class.dbXML.php*. It contains three variables (*$db_name*, *$db_user*, and *$db_pass*) and four functions (*add*, *getdata_xml*, *delete*, and *update*) plus a constructor. The *add* and *update* functions both take a string naming the table, an XML string with the changes or additions to be made and a modify variable to return any errors. The function parses the XML string and builds the query string, then preforms the query and returns true if successful. The important bit is that the XML string must be in a specific format which is as follows:

Add XML String Format

```
<root>
  <newdata>
    <column1>value1</column1>
    <column2>value2</column2>
    .
    .
    <columnN>valueN</columnN>
  </newdata>
</root>
```

As you can see there is a root node followed by a newdata node. The newdata node tells the function that the children of that node will be added to the table. Each child's name is the name of the column and its value is the value to be added. Its quite simple and efective as the xml standard is common in many languages. To create the XML strings themselves I use PHP's built in XML DOM (Document Object Model).

Update XML String Format

```
<root>
  <criteria>
    <column>value</column>
  </criteria>
  <newdata>
    <column1>value1</column1>
```

```
    <column2>value2</column2>
    .
    .

      <columnN>valueN</columnN>
  </newdata>
</root>
```

Here you can see that the root node has two children, criteria and newdata. Newdata is the same as it was in the add string except that the values represent the new value of an existing column. The criteria node contains the column name and value that identifies the row to be updated (usually <id>2</id> where 2 is the id of the table row). Using this method you first must create a *domDocument* object. Then you use a series of createElement() and appendChild() functions to build the domDocument. Finally you can save the object as an XML string or save it as an XML document. To demonstr*ate* this there is my *get_add_string()* function which takes an associative array and converts it to an XML string.

```
function get_add_string($newdata)
      {
      //create the xml string
      $doc = new domDocument('1.0');
      $root = $doc->createElement('root');
      $root = $doc->appendChild($root);
      // add nodes for newdata
      $new = $doc->createElement('newdata');
      $new = $root->appendChild($new);
      foreach ($newdata as $fieldname => $fieldvalue)
            {
            $child = $doc->createElement($fieldname);
            $child = $new->appendChild($child);
            $value = $doc->createTextNode($fieldvalue);
            $value = $child->appendChild($value);
            }
      $xmlString = $doc->saveXML();
      return ($xmlString);
      }
```

The *getdata_xml* and *delete* functions both take a query string and the same *$dberror* variable. The *delete* function just performs the query and returns true on success. The *getdata_xml* function however runs the query and then builds an XML string to pass the results back. The format of the returned string is as follows:

Get Data XML Format

```
<root>
  <row>
    <column1>value1</column1>
    .
    <columnN>valueN</columnN>
  </row>
  <row>
    <column1>value1</column1>
    .
    .
    <columnN>valueN</columnN>
  </row>
  .
</root>
```

For each row that the query returns the function creates a new node named *row*. Within each *row* node, are the column and values of the table row. Along with the *getdata_xml* function I often pair it with the built in simplexml_load_string() function. This function came in vary usefull in my website. It takes an XML string and converts it to an object. With the object I can loop threw the results vary easily to get the data out. For an example of this series take a look at this vary common series of code in my site:

```
$data_list_xml = $db->getdata_xml("some select statement", $dberror);
$data_list = simplexml_load_string($data_list_xml);
foreach($data_list->row as $data) {
  print $data->id;
  ect...
}
```

To create the XML strings for update and add I create two functions (get_add_string and get_edit_string) located in /elements/php/dbfunctions.php. Now you may ask why not just skip the whole XML step and pass the db class the associative array instead. Well the answer is that I wanted to learn how to parse and create XML documents, something before the project I knew nothing about. This method also has some advantages that I didn't really get into for this project. First it would be easy to preform multiple database operations by building one large XML string with many adds and updates then pass it to the db class. This would save bandwidth and server CPU time by not having to keep connecting and disconnecting from the database for each operations. Also since XML strings can easily be saved to XML files you could store a record of all database operations performed on the sever. And not to mention that XML strings are easily parsed by Javascript as well as PHP and many other languages where as associative arrays are not. This method leaves the possibility of combing PHP and Javascript to create cool and useful AJAX applications. I am happy that I learned the DOM and XML in this project so that in the future I can use those skills in other applications.

PDFs and Graphs

One feature that we wanted from the beginning was a nice way to view the results of the evaluation including graphs and PDFs for easy printing and dispersement. I started by creating a table and graph view as I had never created a PDF in any programming language prior to this project and didnt know where to start. I was given an example Excel sheet to base my HTML table off of. There is a table of

Kari Stromberg, Results		Program Average Fall 2006	Individual Average Fall 2006	Program Average for Section	Individual Average for Section
Teaching Assistant Role and Relationship				**3.3333**	**3.5000**
1	I understand the role of the TA in the class.	2.0000	2.0000		
2	My TA Knows my name.	3.0000	3.0000		
3	My TA is approachable in class and out of class.	3.0000	3.0000		
4	My TA communicates with me outside of class.	4.0000	4.0000		
5	My TA is easy to reach when I have a question or concern.	4.0000	4.0000		
6	My TA listens and tries to understand me.	4.0000	4.0000		
7	My TA responds respectfully to my questions and ideas.	5.0000	5.0000		
8	I would be willing to share personal issues with my TA.	3.0000	3.0000		
Class Involvement				**4.2857**	**4.3333**
9	My TA and instructor work well as a teaching team.	4.0000	4.0000		
10	My TA is clear and organized when giving presentations.	4.0000	4.0000		

Illustration 5: Evaluation Results Table

results for each TA in the database. Each table includes program and individual averages for each question as well as each section, and then total averages for the program and the individual TA. My *loadTable.php* and get*Graph.php* files have some of the more complicated query strings in the whole site due to the complicated nature of the table.

I then created a dynamic bar graph to represent the results. Before this project I have had vary little experience with the PHP Image functions. So I began with a simple image and built on it until I had a vary nice looking bar graph that would gracefully expand or contract without breaking as more questions and headers and added or removed. I managed to display all the relevant data from the table in the bar graph.

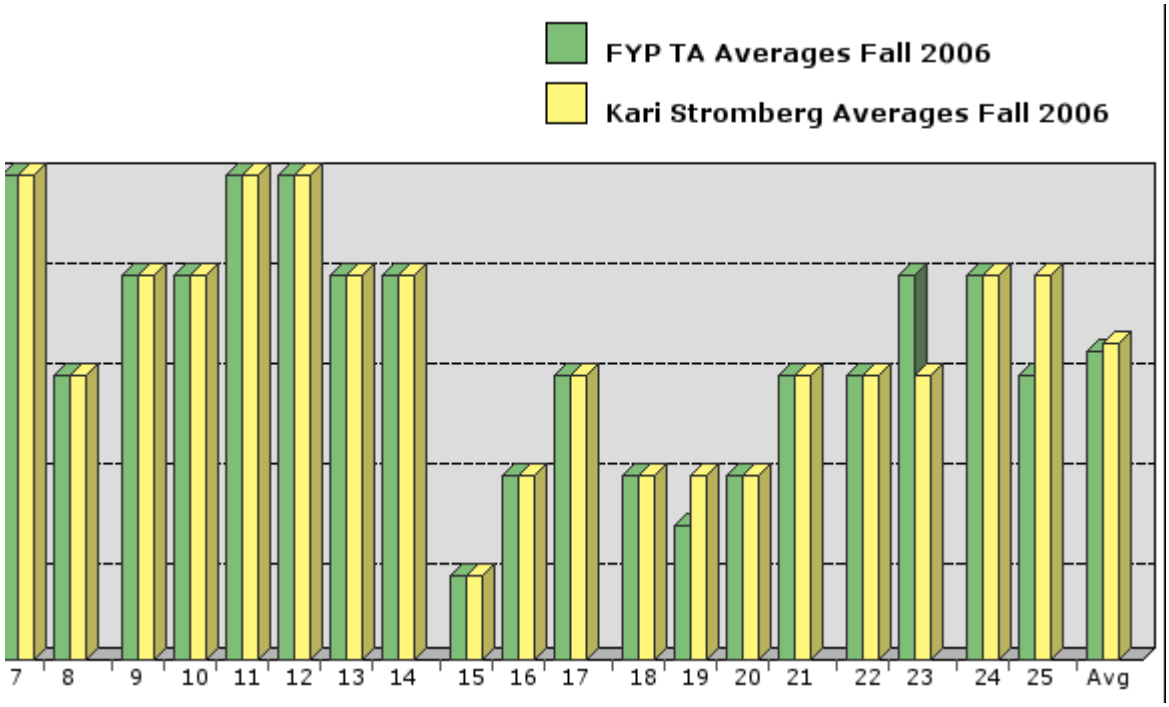

Illustration 6: Evaluation Results Graph

The final step for the results was to create a PDF. From what I initially read it would not be simple to convert a complex table with dynamic data into a PDF. I used the fPDF library to help in the process and it turned out to be not so hard. The tricky part was getting the multi-line table cells to display correctly. The trick was to grab the x and y coordinates before you print the multi-line cell then manually set the position where it is supposed to be after you print the cell. Other than that I just had to simplify my HTML table design a bit for the PDF version but it still looks good and displays all

the information.

Navigation Bar

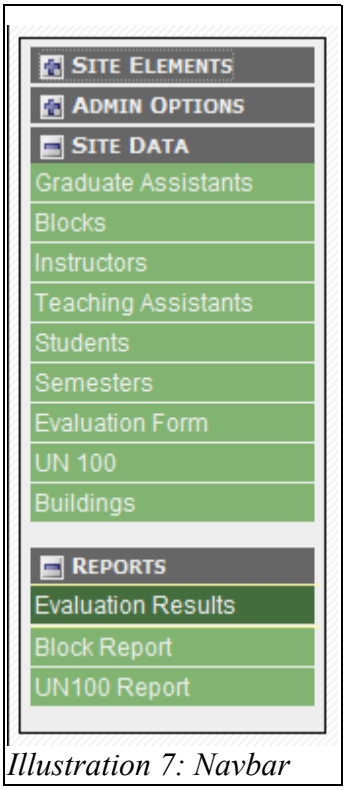

Illustration 7: Navbar

The Navbar is the section to the left of the content on then main administration page. The way I made it so it was automatically generated form the database was one of the trickiest parts of this project. As I said earlier I was tired of having the navbar code floating around in a bunch of different PHP files that had to all be changed when I added a new feature or needed to rearrange the links. So my solution was to put the navbar in the database and use just one file to draw the navbar. As you can see in *Illustration 7* the navbar contains multiple headings which are collapsible hiding or showing the links below that heading. Each heading has a specific user role tied to it so all the links under a heading are also tied to that user role. For example, the Graduate Assistant can not see any of the links under the *Admin Options* heading and only the Web Master can see the links under the *Site Elements* heading. Under the Site Elements heading is a link called Navigation Bar which lets the Web Master control all aspects of how the navigation bar is displayed (see *Illustration 8*). From there you can change a links name, source, heading or position in the bar. You can edit a heading's

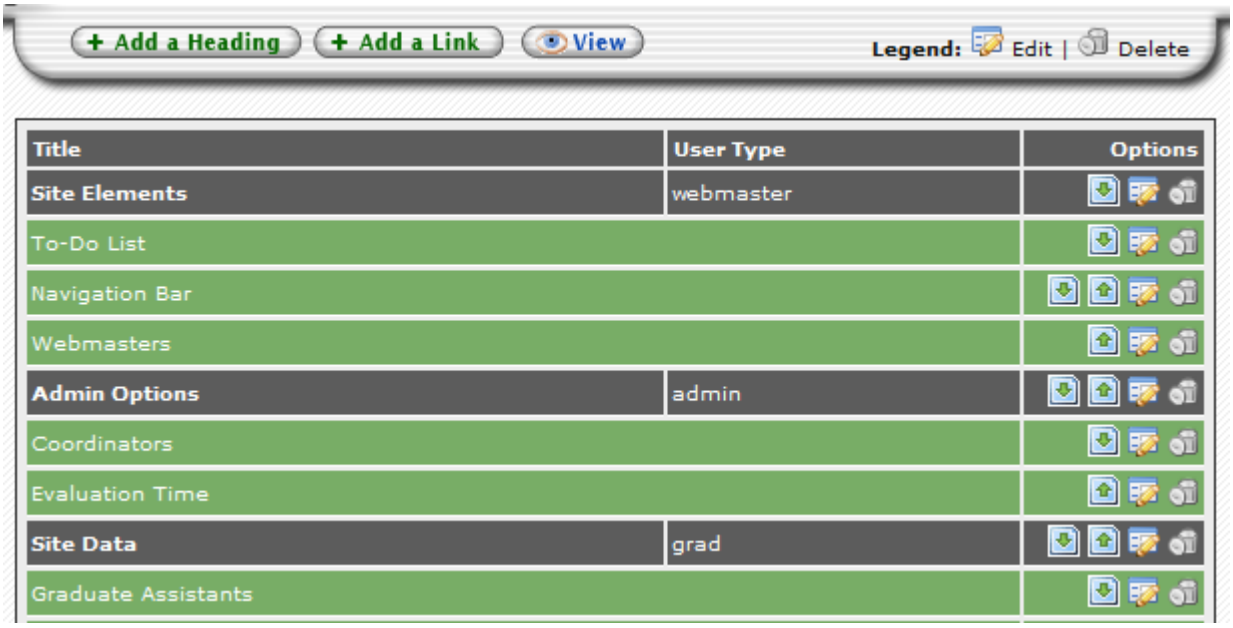

Illustration 8: Navigation Bar Page

name, user role or position. Also you can delete or add a heading or link. In order to get the positioning right, I created a function called *moveLink()* which either swaps the link with the one above it or the one below it. When a link is deleted it first must be moved to the bottom of the list as to not disrupt the positioning of the other links. This is the same for the headings. Deleting or moving a heading will delete or move all links below it. Moving the navigation bar controls to the database was a great time saver for me after I got it finished and I think it is a great addition to making the site easier to manage by a person that knows little about HTML and PHP.

User Authentication and Sessions

Sessions play an important role in the Teacher Evaluation site. In addition to making sure a user is logged in and is in the correct role to view a certain section of the site, sessions allow me to pass important information about the user and his/her actions on the site to different pages where that information is needed. When a user loges in, his/her user name and password is posted to the *auth.php* file located in /evaluation/. This file includes the *sess* class whose job is to check the password with the database and register the session variables. I use *crypt()* to encrypt and compare the passwords in the database for complete password anonymity. If the password is correct the *auth.php* file asks the *sess* object to build and return the values of the variables we want to register in the session. Currently it registers the user's *id*, *fname*, *lname* and *type* as I will likely need to use those values later and often in the site. After the sessions are registered, *auth.php* redirects the user to the correct starting page as described by the user's *type*. Sessions are also used to pass variables back in forth between files in operations such as getting the form posts from the options bar in the Students and Results sources. Sessions allow the website experience to be more seamless like an application by remembering important information while spanning multiple web pages. This is the first website that I needed to use sessions instead of cookies and I think it worked out great.

CSS Design and Browser Compatibility

My goal with the CSS design of the site was to make it clean, professional and improve usability of the site without getting in the way of the user and the job he/she needs to do. I came up with a design whit four main elements, Navigation/Links Bar on the left, content to the right, an options bar that is always visible above the content and a header with more option fixed positioned on the top right of the page. The Navigation bar is positioned in a place that is always visible and easy to access. without

taking up too much screen space. The Header is fixed to the top left of the page with a z-index (css tag describing the depth of an element) that allows the content to scroll under it. The options bar located at the bottom of the header just above the content is used to offer specific options for each source or link. Also statically positioned on the top right of the header are where global options are placed like login/logout and select semester. The Content is set to a fixed width so that the page will look great on 800x600 screens up to 1280x1024.

With a fairly complicated CSS design you can expect to run into trouble getting it to play nice with the different browsers, specifically the abysmal Internet Explorer 6 which is used by over 70% of Internet users. The main problems I ran into are that IE6 doesn't support transparent PNGs or static positioning. To solve this I used the !important CSS trick that makes any tag followed by !important invisible to IE but visible to Firefox and other browsers. This way when an IE6 browser reached my page it gets a GIF header instead of the PNG for all other browsers including IE7. The site however falls apart in Opera which baffles me because the site is CSS compliant and Opera in well know for having great CSS support since it was developed by the same people that started CSS.

Conclusion

With this project I wanted to create an easy to use and well structured website that could be managed with a web interface while learning as many new techniques and skills as I could along the way. I learned all about XML which I think is the the present and future of the web. I can now parse and create valid XML documents with ease and use them in cool new ways. Weather my technique of passing XML strings back in forth is productive or not it does leave open the possibilities for expansion into AJAX in the future. I learned the process of creating a large scale website from scratch and I'm sure learning from my mistakes will make it go smoother in the future. It is vary important to create a well structured database before beginning the coding as I learned by having to redo a lot of code after redesigning the database half way threw the project. And having a well organized file structure saves a lot of time later on in the programming process. I found creating the graphs and pdfs some of the most fun challenges of this project. The creating of the PDF however was not as difficult as I originally presumed thanks to the fpdf library and the helpful documentation.

When starting this project I was excited by all the new and emerging web application in the Web 2.0 boom of 2006. There are many limitation in developing for the web and it feels new to me as if there are still so many undiscovered solutions to be found. I think this is one of the reasons I am drawn to the Internet as a media for my programming. To me it feels like almost every piece of software I can think of writing has already been done ten times over and better then I can do it anyways. With the constant evolution of web applications like AJAX, XML, RSS, XHTML, CSS, exciting new PHP frameworks, and the rest of the "acronym soup" of web technologies to keep up with, a web developer needs to stay on his/her toes so they don't get left in the dust.