Bay Audio Repair Website & Data Management Application

by
Michael Shelley

Computer Science Department
College of Engineering
California Polytechnic State University
2010

Date Submitted: 3-16-10
Advisor: Franz Kurfess
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>II.</td>
<td>Related Systems</td>
<td>2</td>
</tr>
<tr>
<td>III.</td>
<td>Requirements</td>
<td>3</td>
</tr>
<tr>
<td>IV.</td>
<td>System Design</td>
<td>29</td>
</tr>
<tr>
<td>V.</td>
<td>Implementation</td>
<td>32</td>
</tr>
<tr>
<td>VI.</td>
<td>Conclusions</td>
<td>39</td>
</tr>
<tr>
<td>VII.</td>
<td>Future Work</td>
<td>39</td>
</tr>
<tr>
<td>VIII.</td>
<td>References</td>
<td>40</td>
</tr>
<tr>
<td>IX.</td>
<td>Appendix</td>
<td>42</td>
</tr>
</tbody>
</table>

ABSTRACT

PAGE

ABSTRACT ........................................................................................................................................ii
ABSTRACT

The goal of this senior project was to build a website and software application to receive and manage audio equipment repair requests for a small startup company called Bay Audio Repair (BAR). Furthermore, it allowed me to gain experience in web development and software engineering practices, specifically requirements gathering, design and implementation. The website provides an online interface for BAR’s customers to request repairs and the application allows BAR employees to update the progress of a repair. Several technologies were used in the system’s construction: HTML, XML, PHP, and C#. 
I. Introduction

Bay Audio Repair (BAR) is a small startup business serving the south Bay Area and specializes in audio equipment repair. In an effort to attract more business BAR has sought development of a website and a software application. The website intends to provide existing and potential customers with company information (e.g. repair policies, repair history, contact info etc.) and an interface for requesting and tracking the status of a repair. The application will only be used internally by BAR to review repair requests and approve or deny them. If a request is approved, BAR technicians can update the status of the repair keeping the customer informed of the repair’s progress.

Problem Statement:

The BAR website and Repair Data Manager application aim to solve the following problems:

1. to provide BAR with a business oriented web presence
2. to provide customers with a convenient method for requesting and tracking the status of a repair
3. to provide company technicians with an easy way of receiving and managing repair information

Personnel:

1. BAR employees
2. BAR customers
3. system developer

BAR employees will use the Repair Data Manager application to receive, assess, update and finalize repair request information. Repair requests are archived in a database, creating a centralized account of the company’s business.

A BAR customer is any person or entity that requests a repair and agrees to BAR’s repair policy.

I, Michael Shelley, am the primary BAR webmaster and developer of the Repair Data
Manager.

**Operational Setting:**

BAR’s website has one operational setting, for use as a normal publicly accessible website serving as an interface between customers and the company. Anyone with a contemporary web browser and an Internet connection can access the site to learn about BAR and/or request a repair. An email account is necessary to request a repair.

The Repair Data Manager application is intended for internal use by BAR employees only.

**Impacts:**

Potential positive impacts of Bay Audio Repair’s website are increased exposure of the company’s services—hopefully translating into higher profits and a smoother means of informing customers about their repair(s). Negative impacts might result if the website or Repair Data Manager application is poorly designed and doesn’t prove beneficial to customers or the company.

Negative results can be avoided if the developer can accurately capture BAR’s functional vision of the software in the requirements stage. It is necessary for the developer and BAR to discuss any misunderstandings or questions as they arise and modify the requirements to ensure a desirable end product.

V. Related Systems

Below are links to websites of other audio repair companies:

http://www.approvedaudioservice.com

http://www.audioelectronics.com/

http://www.atlasaudiorepair.com/
All of these websites have sections in common with one another. Namely, an overview of the company, services offered, repair policy & rates, FAQ and contact information. BAR’s website aims to have similar sections with the important addition of a repair request web form and the ability for customers to check a repair’s progress.

Existing task tracking applications include:

http://www.jestertool.com/

http://www.vip-qualitysoft.com/products/organizer/task_tracking_software/

http://www.repairwizard.com/

According to the manufacturers’ websites all the above products are capable of general task tracking. However, it is uncertain whether they are configurable to interact with a remote data source. Repair Wizard, specifically targeted at small repair businesses, seems to have characteristics closest to those desired by BAR’s repair data management application. An excerpt from Repair Wizard’s website describing their “Service Tracker” states:

The Service Tracker helps you track the repair work through your shop making it easier for you to manage your work flow.

Service Tracker generates reports such as:

- Units In Shop
- Items Awaiting Parts
- Items Awaiting Delivery
- Units Received, Completed & Shipped

V. Requirements

The functional requirements are broken up into two areas: the website’s content and the
Repair Data Manager’s features. Most of the website’s sections contain static information about the business. The most interesting aspects of the site are the customer repair request form and a customer’s access to their repair’s progress. The Repair Data Manager is only accessible by the BAR employees and is used to review requests as they arrive and update their statuses over the course of repair. The requirements cover a typical user scenario where a customer requests a repair through the website, a BAR employee reviews the initial request data and approves it through the Repair Data Manager application, the customer completes step 2 of the request process, the customer checks the status of their repair and a repair record is modified to reflect the progress of the repair.

**Bar Website Requirements:**

Bay Audio Repair’s website format is similar to other small business websites frequently found on the Internet. It serves as a useful interface between customers and the company by providing easy access to information about Bay Audio Repair’s services, contact information, repair costs and the progress of a repair. Sections of the website can be classified into three categories:

1. Company Info – About Us, FAQ, Contact Us
2. Repair Related – Repair Policy, Request Repair, Repair Status, Past Repairs
3. Sales Related – For Sale, Sell to Us

The sections are described below in the order they appear on the website.

**About Us**

About Us is the first page seen by the website’s visitors. It contains an advertisement attempting to convince a visitor Bay Audio Repair is the right choice for resolving their audio equipment’s troubles. It also provides a brief synopsis about the company’s services and history.
Repair Policy

The Repair Policy is a detailed statement of customer shipping obligations and expectations, the repair process, service fees and warranty information.

Request Repair

This section contains input fields to gather information about the customer and their equipment’s troubles. Requesting a repair is a two-step process requiring the completion of two HTML forms. A requester first provides their email address, their equipment’s brand and model and its problems/symptoms. The requester is given a unique repair id to complete step two if their request is accepted and to check the status of their repair. A BAR employee uses the Repair Data Manager application to review the initial request and decides whether a successful repair is feasible. If so, the requester is notified by email to complete step two by filling out another form with their name, address, phone number, any accessories necessary for the equipment’s operation, other notes of interest, and the equipment’s insurance value.

Repair Status

Repair status contains an input field for a customer’s unique repair id given to them at the time of their initial repair request (step 1) and returns a report containing all the data associated with the given repair id.

Past Repairs

Repairs completed in the past are archived in this section. Customers can get an idea about the types and brands of equipment the company has previously serviced.
For Sale

This page presents a tabular listing of equipment BAR currently has on sale. A row includes a thumbnail of the product, manufacturer, model and asking price.

Sell to Us

A form for visitors to send BAR a description about any equipment they think the BAR might consider buying. Contains text input fields for the visitor’s email, equipment’s make, model, price and an optional photo.

FAQ

Frequently asked questions. Answers to questions often asked by customers are collected here for future reference.

Contact Us

Presents the visitor with a form for which they can send BAR a question.

2. Repair Data Manager Requirements:

2.1 Repair Request (Step 1)

Beginning at BAR’s homepage the user clicks the link “Request Repair” from the navigation menu on the left-hand side.
Figure 2.1.1: BAR’s Homepage

The user is taken to the page shown in Figure 2.1.2. The user clicks on “Request Repair (Step 1).”
Figure 2.1.2: Request Repair (Step 1)

The user is taken to the page shown in Figure 2.1.3. The user fills out the HTML form with the information asked for. The user clicks the “Submit Request” button to send their request to BAR.
**Figure 2.1.3:** Submit Request

The user is taken to the page shown in Figure 2.1.4 and given a unique repair id.
Step 1 of the repair request is now complete.

2.2 Request Approval

The user executes the Repair Data Manager application. During the application’s loading time it retrieves the file containing new requests from the web server. The application presents the data gathered in “Repair Request Step 1” in the form of a table contained in the “New Requests” tab. The user clicks the “Approve/Deny Request” button.
Figure 2.2.1: Approve/Deny Request Button

The system displays the “New Repair Request” window.
Figure 2.2.2: New Repair Request

The user clicks the “Approve” button. The “New Repair Request” window closes. The row corresponding to the approved request is transferred to the table in the “Active Repairs” tab.
Beginning at BAR’s homepage, the user clicks the “Request Repair” link from the navigation menu on the left-hand side (see Figure 2.2.1). The user is taken to the page shown in Figure 2.3.1. The user enters their repair id in the input box and clicks the “Step 2” button.

**Figure 2.2.3: Active Repairs**

### 2.3 Repair Request (Step 2)

Beginning at BAR’s homepage, the user clicks the “Request Repair” link from the navigation menu on the left-hand side (see Figure 2.2.1). The user is taken to the page shown in Figure 2.3.1. The user enters their repair id in the input box and clicks the “Step 2” button.
The user is taken to the page shown in Figure 2.3.2. The user fills out the HTML form with the information asked for. The user clicks the “Submit” button to complete step 2 of the request process.
Figure 2.3.2: Request Repair (Step 2) Form

The user is taken to the page shown in Figure 2.3.3.
Step 2 of the repair request process is complete.

2.4 Edit Diagnosis, Estimate Fields

The user selects the repair record row corresponding to the repair they wish to edit from the active repairs table in the “Active Repairs” tab of the Repair Data Manager and clicks the “Edit Repair” button.
Figure 2.4.1: Edit Repair Button

The system displays the “Edit Repair” window. The user clicks the “Diagnosis” tab.
Figure 2.4.2: Edit Repair

The user enters the diagnosis in the text area within the “Diagnosis” tab. To enter an estimate the user clicks the “Billing” tab.
Figure 2.4.3: Billing Tab

The system displays the contents of the “Billing” tab and the user enters the dollar amount of their estimate in the “Estimate” textbox.
Figure 2.4.4: Edit Estimate

To save the changes the user has made the user clicks the “Save Changes” button.
Figure 2.4.5: Save Changes Button

The web server receives the new values for the fields edited by the user and modifies the remote data accordingly.
2.5 Check Repair Status, Accept Repair Costs

Beginning at BAR’s homepage the user clicks the “Repair Status” link from the navigation menu on the left-hand side.

Figure 2.5.1: Repair Status Link
The user is taken to the page shown in Figure 2.5.2. The user enters their repair id and clicks the “View Status” button.

![Check Repair Status](image)

**Figure 2.5.2: Check Repair Status**

The user is taken to the page shown in Figure 2.5.3. The last line in the “Billing” section asks the user if they accept the estimated cost of repair. The user clicks the “Yes” button.
Figure 2.5.3: Repair Status

The repair record’s status is updated to reflect the user’s choice of accepting the estimated repair costs.
2.6 Edit Parts and Labor Costs

The user opens the “Edit Repair” window for the repair they wish to edit. The user selects the “Billing Tab” and enters the dollar amount in the labor textbox. The total repair cost is updated.

![Edit Labor Cost](image)

**Figure 2.6.1:** Edit Labor Cost
The user clicks the “Add Part” button. The system displays the “Add Part” window. The user enters the part’s name, quantity and price.

![Add Part Window](image)

*Figure 2.6.2: Add Part*

The “Add Part” window closes and a new row is created in the “Parts” table for the newly added part. The total cost is updated.
2.7 Edit Base URL

In order to access repair data files the Repair Data Manager expects a pre-defined directory structure to exist on the web server. The root of this directory structure is configured through the “Edit” drop down menu.

Figure 2.6.3: Parts Table
The user clicks the “Edit” drop down menu.

![Edit Menu](image)

**Figure 2.7.1: Edit Menu**

The user clicks “Base URL” from the drop down menu. The system displays the “Edit Base URL” window. The user enters the address of the website’s root.
IV. System Design

Overall Architecture:
Website Design:

BAR’s website is similar to many websites encountered on the web. It has three main components as shown in the figure below.

Bay Audio Repair Masthead

Navigation Menu

link 1
link 2
.
.
link n

Content displayed here depending on which link a visitor chooses from the Navigation Menu.
Repair Data Manager Class Diagrams:

```
RepairDataManager
Dictionary<string, string> LOCAL_PATHS
RepairDataClient rdc
Table newRequests
Table activeRepairs
populateNewRequestsTable()
populateActiveRepairsTable()
refreshRequestTable()
showApproveDenyView()
showEditRepairView()

RepairDataClient
Dictionary<string, string> REMOTE_PATHS
WebClient webClient
getRepairFile()
approveDenyRepair()
updateRepairFile()
```

```
ApproveDenyView
TextBox repairID
TextBox date
TextBox email
TextBox brand
TextBox model
TextArea problem
getRepairID()
getDate()
getEmail()
getBrand()
getModel()
getProblem()
approve()
deny()

EditRepairView
TextBox name
TextBox quantity
TextBox cost
addPart()
```
V. Implementation

Website:

Before this project I had limited knowledge and experience in web development. To better understand how to build a website I spent two weeks researching different approaches. Despite the existence of many widely used website builder applications such as Dreamweaver, Joomla, Frontpage, etc., I knew I wanted to build the BAR website by hand using a plain text
editor. Doing so would give me a sense of what one of those applications does behind the scenes as a user arranges content and allow me greater control and experimental power.

I discovered there are two main approaches to organizing the content of a page, tables or HTML <div> tags. After reading an article on the website smashingmagazine.com titled, "Table Layouts vs. Div Layouts: From Hell to...Hell," I concluded a div layout would take the least time to develop. To find examples of div based sites I searched around looking at the HTML and CSS source of different sites. I stumbled across http://www.bvlacrosse.com/ and found its source understandable and well structured. This page along with another, http://www.bigbaer.com/, served as my inspiration and reference while piecing together BAR's website.

It took some tinkering with different CSS properties to achieve the correct alignment for the components of the website. Three <div>s make up the site's core sections as shown in the Website Design. I tried to produce a site with a decent appearance and a logical flow. If I had more time I would try to embellish it a little more. For example, a more attractive navigation menu could be developed using jQuery.

Below is a figure showing the directory hierarchy of the website.
Each of the second tier directories correspond to one of the links from the navigation menu. Each contains an index.html file that is served when a visitor navigates to that section of the website. If the page contains an input form, the directory also contains a script to process the input.

I chose PHP as the scripting language to process user input sent through the various HTML forms. When a visitor makes a new repair request, their input is passed along to a script in the "new_request" directory. PHP allows easy access to the data through its "superglobal" $_POST array. The data is entered as a new entry into the "new_requests.xml" file in the same directory. An example of "new_requests.xml" with two new repair requests is shown below.

```xml
<?xml version="1.0"?><repair_requests>
  <repair id="4b973ca2c118">
    <req_date>03-09-10</req_date>
    <email>mrfoo@example.net</email>
    <brand>Fender</brand>
    <model>Guitar Pedal 1A</model>
    <problem>No output!</problem>
  </repair>
  <repair id="4b973cee9e4cc">
    <req_date>03-09-10</req_date>
    <email>bizbaz@example.net</email>
    <brand>Marshall</brand>
    <model>amplifier-2000</model>
    <problem>worn out knobs</problem>
  </repair>
</repair_requests>
```

After a repair request is approved its entry in "new_requests.xml" is transferred to "active_repairs.xml." The repair_records directory contains XML files corresponding to specific repair requests. A new file is created in repair_records after the customer has completed step 2 of the request process. Each file follows the naming convention "repair_id.xml" (e.g. 4b81c4d66a606.xml). Below is an example directory listing of the repair_records directory:

```
mrshelle@vogon:-/www/sp/bar/request_repair $ ls
active_repairs/repair_records/
```
Here is a look inside one of these repair records:

```xml
<?xml version="1.0"?>
<repair>
  <repair_data>
    <serial>75644422</serial>
    <accessories/>
    <notes/>
    <diagnosis>Three bad capacitors in the power circuit. Bad 5v transformer.</diagnosis>
    <fix>Replace capacitors/transformer.</fix>
  </repair_data>
  <customer>
    <first>Fred</first>
    <last>Oobar</last>
    <phone>805-555-5555</phone>
    <street>1024 Magnolia Lane</street>
    <city>Los Osos</city>
    <state>CA</state>
    <zip>93401</zip>
  </customer>
  <billing>
    <estimate>50.00</estimate>
    <parts>
      <part quantity="1" price="$5.00">5v transformer</part>
      <part quantity="3" price="$0.50">10mF capacitor</part>
    </parts>
    <labor>45</labor>
    <billed>2-20-09</billed>
    <paid>3-1-10</paid>
  </billing>
  <shipping>
    <received>2-15-10</received>
    <returned>3-3-10</returned>
    <carrier>UPS</carrier>
    <insurance>$35</insurance>
    <tracking_num>23121131</tracking_num>
  </shipping>
</repair>
```

When a user checks the status of their repair a script checks to see if the repair id they entered exists in the "active_repairs.xml" file. If it exists, a script echoes the entry in
"active_repairs.xml" corresponding to the given repair id and also the contents of its
repair_record file. Below is a snippet of the PHP code that performs this operation:

```php
$actRepsDOM = new DOMDocument();
$actRepsDOM->load('../request_repair/active_repairs/active_repairs.xml');
$repairs = $actRepsDOM->getElementsByTagName('repair');

if((($repair = findRepair($id, $requests)) != NULL) ||
((repair = findRepair($id, $repairs)) != NULL)) {
    $status = $repair->getAttribute('status');
    $req_date = $repair->getElementsByTagName('req_date')->item(0)->nodeValue;
    $email = $repair->getElementsByTagName('email')->item(0)->nodeValue;
    $brand = $repair->getElementsByTagName('brand')->item(0)->nodeValue;
    $model = $repair->getElementsByTagName('model')->item(0)->nodeValue;
    $problem = $repair->getElementsByTagName('problem')->item(0)->nodeValue;

    if($status == NULL) {
        $status = 'pending review';
        $type = 'New Request Summary';
    } else {
        $type = 'Active Repair Summary';
    }

    echo "<fieldset style='max-width:500px;'>" .
        "<legend>" .
            "<b><font color='white'>" . $type . "</font></b>" .
        "</legend>" .
        "<p>Status: <font color='red'>" . $status . "</font></p>" .
        "<p>Request Date: " . $req_date . "</p>" .
        "<p>Email: " . $email . "</p>" .
    "</fieldset>" .
```

**Repair Data Manager:**

On account of its large and well documented library, I chose Visual C# as the language to
build the Repair Data Manager. Once I learned my way around the Visual Studio IDE creating
the GUI components was straightforward. Developing the underlying code to deal with the
repair data was a more time consuming task. It required retrieving the data from the web server,
extracting it from its XML format, populating the GUI with it and sending any actions/changes the user performed on it back to the remote data files.

When the Repair Data Manager loads it retrieves the "new_requests.xml" file from the web server. The retrieval is performed using the WebClient class. In network terms the Repair Data Manager uses HTTP as its "application protocol." To save data repair changes made by a user the application uses the HttpWebRequest class. It allows the application to send HTTP POSTs to remote scripts. For example when a user approves a repair request the application sends a POST to a script named "approve.php" residing in the active_repairs directory of the website. The script processes the post data to execute the correct action, in this case to transfer the entry associated with the given repair id in "new_requests.xml" to "active_repairs.xml."

Here is a look at the postData function:

```csharp
/**
 * Setup the WebRequest object and send the HTML POST to the remote
 * script.
 * 'data' the data being POSTed.
 * 'remotePath' indicates the remote script to handle the POST
 */
private void postData(byte[] data, string remotePath)
{
    /* C#'s class to handle HTTP POST transmissions. */
    HttpWebRequest WebReq = (HttpWebRequest)WebRequest.Create(remotePath);

    /* All remote scripts are set to accept HTML POST data */
    WebReq.Method = "POST";

    /* Our communication is analogous to a HTML form with its method=POST
    and the action set to a script to process the POST variables. */
    WebReq.ContentType = "application/x-www-form-urlencoded";

    /* data's length. */
    WebReq.ContentLength = data.Length;

    /* Stream to write POST data too. */
    Stream PostData = WebReq.GetRequestStream();

    /* Write the data to the stream, close the stream. */
    ```
PostData.Write(data, 0, data.Length);
PostData.Close();

/* Get the POST's response, would probably be useful to detect errors here and report them to the user if any occur. */
HttpWebResponse webResp = (HttpWebResponse)WebReq.GetResponse();

/* Print the status code of the POST's response */
Debug.WriteLine(webResp.StatusCode);
Debug.WriteLine(webResp.Server);

I used two different ways of dealing with the XML data. When populating the tables of the "New Requests" and "Active Repairs" tabs a Language-Integrated Query (LINQ) was used. The LINQ documentation describes LINQ as, "`.NET Language-Integrated Query defines a set of general purpose standard query operators that allow traversal, filter, and projection operations to be expressed in a direct yet declarative way in any .NET-based programming language.""

Below is the LINQ query used for the "New Requests" table (repairs is an instance of XDocument-allows LINQ queries, newReqsDGV is the table object):

```csharp
var query = from newReqs in repairs.Descendants("repair")
            select new
            {
                RequestID = newReqs.Attribute("id").Value,
                RequestDate = newReqs.Element("req_date").Value,
                Email = newReqs.Element("email").Value,
                Brand = newReqs.Element("brand").Value,
                Model = newReqs.Element("model").Value,
                Problem = newReqs.Element("problem").Value
            };
newReqsDGV.DataSource = query.ToList();
```

To populate the EditRepair class's Textboxes and parts table the XmlDocument class was used. XmlDocument represents a XML file using the Document Object Model (DOM).

Below is a code snippet showing how a few of the Textboxes are populated:

```csharp
/**
 * Populate Customer Info, Billing and Shipping tabs' fields.
 */
```
private void populateTxtFields(string repairFile)
{
    repairRec = new XmlDocument();
    repairRec.Load(repairFile);
    XmlNodeList children =
        (repairRec.GetElementsByTagName("repair_data"))[0].ChildNodes;

    this.serial.Text = children[0].InnerText;
    this.accessories.Text = children[1].InnerText;
    this.notes.Text = children[2].InnerText;
    this.diagnosis.Text = children[3].InnerText;
    this.fix.Text = children[4].InnerText;

    VI. Conclusions

    At the outset of the project I had a good discussion with BAR's owner about the
    functional requirements of the system, but as time went on our communication waned. I would
    have liked to have gotten more feedback during different stages of development.

    A few features are still missing from the system. For example, the initial specification
called for sending notification emails to customers when the status of their repair changed. PHP
provides an email function, but I didn't have enough time to integrate all the calls in. I also left
out the feature to deal with repairs under warranty.

    I believe I accomplished the project's most significant objective: to build an interactive
website and data management application. The project also allowed me to familiarize myself
with technologies I had little or no prior experience with; namely HTML, CSS, PHP, XML, C#
and the Visual Studio IDE. I feel the current version is satisfactory, but some refactoring and
further development could increase the system's robustness and usefulness.

    VII. Future Work

    Data Validation:
I didn't have time to research techniques for scrubbing user input submitted through HTML forms. I did find a PHP function that validates email addresses, but the version of PHP on vogon.csc.calpoly.edu at the time of this writing doesn't support that function. I read a little into XML schemas and found using one for the repair data files might be a good way of ensuring the files' element values are of the correct type. Adding a "captcha" object to the web forms might help avoid spam.

**Error Checking:**

Similarly, I didn't perform a lot of error checking in the Repair Data Manager, runtime exceptions still occur in certain scenarios. Before the application could be deployed in the real world it would be wise to harden it.

**Sorting:**

It would be nice to allow a user to sort the data in the tables of the Repair Data Manager by different columns. Adding color coding for different repair "status" values would be another useful enhancement.

**Refresh/Save Changes Progress**

When the Repair Data Manager is retrieving or saving to a remote file it would be nice to display a progress bar or a notification that the operation is still occurring.

VIII. References
[1] Wavik, Geir "Table Layouts vs. Div Layouts: From Hell to...Hell?"

[2] -Website source reference-
http://www.bvlacrosse.com/

[3] -Website design reference-
http://www.bigbaer.com/

[4] "How to use tables to structurize forms in HTML and about alternatives, like fieldset"
[Online Site] [2009 December 1], Available at HTTP: http://www.cs.tut.fi/~jkorpela/forms/tables.html

[5] "HTML Forms: Form Fields"
[Online Site] [2009 November 20], Available at HTTP: http://www.echoecho.com/htmlforms04.htm


[7] Herrington, Jack "Reading and writing the XML DOM with PHP"

[8] Megginson, David "Ditching the Database: XML and the PHP web app"
[Online Site] (2006 May 16) [2010 January 7], Available at HTTP: http://xtech06.usefulinc.com/schedule/paper/19

[9] Liberty, Jesse "Learning C#"
O'Reilly, 2002.

[10] ".NET Framework Class Library"
[Online Site] [2010 January 15], Available at HTTP:
IX. Appendix

**BARRepairManager.cs**

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace BAR_Repair_Manager
{
    public class BarRepairManager
    {
        /**
         * Main method that launches the RepairDataManager Form.
         */
        public static void Main()
        {
        }
    }
}
```
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new RepairDataManager());

RepairDataManager.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Xml.Linq;
using System.Text;
using System.Windows.Forms;

namespace BAR_Repair_Manager
{
    /*
     * Top level form for repair inventory management. Houses a tab control
     * with three tabs. Each tab contains a DataGridView control to display
     * newly requested repairs, active repairs and repairs under warranty.
     * Repair data is accessed through the RepairDataClient class and used to
     * populate the tabs' DataGridViews.
     */
    public partial class RepairDataManager : Form
    {
        /* Stores local paths to data files, initialized in constructor */
        public static readonly Dictionary<string, string> LOCAL_PATHS =
            new Dictionary<string, string>();

        private readonly string NEW_REQUESTS = "NEW_REQUESTS";
        private readonly string ACTIVE_REPAIRS = "ACTIVE_REPAIRS";
        private readonly string REPAIR_RECORDS = "REPAIR_RECORDS";

        /* Accesses remote xml data repair files. */
        private RepairDataClient rdc;

        public RepairDataManager()
        {
            InitializeComponent();
            newReqsDGV.MultiSelect = false;
            actRepsDGV.MultiSelect = false;

            LOCAL_PATHS.Add(NEW_REQUESTS, @"temp\new_requests.xml");
            LOCAL_PATHS.Add(ACTIVE_REPAIRS, @"temp\active_repairs.xml");
            LOCAL_PATHS.Add(REPAIR_RECORDS, @"temp\repair_records\");
rdc = new RepairDataClient();
}
/**
 * Uses LINQ to query a xml repair data file and sets a tab's
 * DataGridView DataSource property to the query results.
 */
private void populateDGV(string pathKey)
{
    /* Get remote xml repair data file. */
    rdc.getXMLFile(pathKey, String.Empty);

    /* Open local xml repair data file for LINQ access */
    XDocument repairs = XDocument.Load(LOCAL_PATHS[pathKey]);
    if (pathKey.CompareTo(NEW_REQUESTS) == 0)
    {
        //populate new repair requests tab's DataGridView
        /* LINQ query to get row values for insertion into
         * New Request tab's DataGridView. */
        var query = from newReqs in repairs.Descendants("repair")
        select new
        {
            RequestID = newReqs.Attribute("id").Value,
            RequestDate = newReqs.Element("req_date").Value,
            Email = newReqs.Element("email").Value,
            Brand = newReqs.Element("brand").Value,
            Model = newReqs.Element("model").Value,
            Problem = newReqs.Element("problem").Value
        };
        newReqsDGV.DataSource = query.ToList();
    }
    else if (pathKey.CompareTo(ACTIVE_REPAIRS) == 0)
    {
        //populate active repair requests tab's DataGridView
        /* LINQ query to get row values for insertion into
         * Active Repair tab's DataGridView. */
        var query = from actReps in repairs.Descendants("repair")
        select new
        {
            RequestID = actReps.Attribute("id").Value,
            RequestDate = actReps.Element("req_date").Value,
            Status = actReps.Attribute("status").Value,
            Email = actReps.Element("email").Value,
            Brand = actReps.Element("brand").Value,
            Model = actReps.Element("model").Value,
            Problem = actReps.Element("problem").Value
        };
        actRepsDGV.DataSource = query.ToList();
    }
}
/**
 * Event handler for new requests tab clicks and "Refresh" button clicks.
 */
private void populateNewReqsDGV(object sender, EventArgs e)
{
    populateDGV(NEW_REQUESTS);
}

/**
 * Event handler for active repairs tab clicks and "Refresh" button clicks.
 */
private void populateActRepsDGV(object sender, EventArgs e)
{
    populateDGV(ACTIVE_REPAIRS);
}

/**
 * Event handler for the Approve/Deny Request button click (New Requests tab).
 * Instantiates and shows the ApproveDenyForm.
 */
private void approveDenyReqBtn_Click(object sender, EventArgs e)
{
    ApproveDenyForm rowForm =
        new ApproveDenyForm(newReqsDGV.SelectedRows[0].Cells, rdc);
    rowForm.OnApproveOrDenyBtn_Click +=
        new ApproveDenyForm.ApproveOrDenyBtnHandler(populateNewReqsDGV);
    rowForm.Show();
}

/**
 * Event handler for the edit drop down menu button click.
 * Instantiates and shows the BaseURLEditor Form.
 */
private void baseURLToolStripMenuItem_Click(object sender, EventArgs e)
{
    new BaseURLEditor().Show();
}

/**
 * Event handler for "Edit Repair" button clicks (Active Repairs tab).
 * Instantiates and shows the Repair Editor form.
 */
private void editRepData_Click(object sender, EventArgs e)
{
    RepairEditor repEdit =
        new RepairEditor(actRepsDGV.SelectedRows[0].Cells, rdc);
    repEdit.Show();
}
/**
 * Refreshes a tab's DataGridView upon tab clicking.
 */
private void repairDataSheets_SelectedIndexChanged(object sender, EventArgs e)
{
    if (this.repairDataSheets.SelectedIndex == 0)
    {
        populateDGV(NEW_REQUESTS);
    }
    else
    {
        populateDGV(ACTIVE_REPAIRS);
    }
}

ApproveDenyForm.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
namespace BAR_Repair_Manager
{
    /****
    * Displays a row from RepairDataManager's New Requests tab's DataGridView, allowing
    * the user to review the new repair request in more detail and make the
decision to
    * approve or deny the request.
    */
    public partial class ApproveDenyForm : Form
    {
        private RepairDataClient rdc;

        /* Event declaration. This event is raised upon approval/denial of a
        repair
        request prompting the New Requests tab's DataGridView to update
        itself. */
        public delegate void ApproveOrDenyBtnHandler(object sender, EventArgs e);
        public event ApproveOrDenyBtnHandler OnApproveOrDenyBtn_Click;
constructor. Populates the form's fields with data from the
selected row of
* New Requests tab's DataGridView.
*/
public ApproveDenyForm(DataGridViewCellCollection newReq,
RepairDataClient rdc)
{
    InitializeComponent();

    /* Populate the Form's textField controls with data from the
selected row
('newReq'). */
    this.Text = "New Repair Request (" + (string)(newReq[2].Value) + ""
    this.reqIDTxt.Text = (string)(newReq[0].Value);
    this.dateTxt.Text = (string)(newReq[1].Value);
    this.emailTxt.Text = (string)(newReq[2].Value);
    this.brandTxt.Text = (string)(newReq[3].Value);
    this.modelTxt.Text = (string)(newReq[4].Value);
    this.problemTxt.Text = (string)(newReq[5].Value);

    this.rdc = rdc;
}

/**
* Event Handler for the "Cancel" button click.
*/
private void reqFormCancelbutt_Click(object sender, EventArgs e)
{
    this.Close();
}

/**
* Event Handler for the "Approve" button click.
*/
private void approveReqBtn_Click(object sender, EventArgs e)
{
    rdc.approveOrDenyRequest(this.reqIDTxt.Text, "approve");
    OnApproveOrDenyBtn_Click(this, new EventArgs());
    this.Close();
}

/**
* Event Handler for the "Approve" button click.
*/
private void denyReqBtn_Click(object sender, EventArgs e)
{
    rdc.approveOrDenyRequest(this.reqIDTxt.Text, "deny");
    OnApproveOrDenyBtn_Click(this, new EventArgs());
    this.Close();
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace BAR_Repair_Manager
{
    /****
     * Presents all the information associated with a repair. Contains four top level
     * TabControls (Repair Data, Customer Info, Billing and Shipping).
     * Allows the user
     * to modify most of the fields (repair id, request date and a few other fields
     * provided by the customer are immutable). This form serves as the business's avenue
     * for updating repair data.
     */
    public partial class RepairEditor : Form
    {
        /'* Styling for parsing monetary decimals */
        public static readonly NumberStyles PRICE_STYLE =
            NumberStyles.AllowCurrencySymbol | NumberStyles.AllowDecimalPoint |
            NumberStyles.AllowThousands;

        private RepairDataClient rdc;
        private XmlDocument repairRec;
        private List<object> chgdFields; //holds user modified fields
        private bool isUserChg = false; //flag indicating if user has modified anything

        private decimal partsTotal; //total parts cost

        /**
         * Constructor. Populates the Form's Repair Data tab with the values of the
         * selected row of RepairDataManager's Active Repair tab's DataGridView.
         */
        public RepairEditor(DataGridViewCellCollection row, RepairDataClient rdc)
InitializeComponent();
setupPartsDGV();
partsTotal = 0.00M; //initialize partsTotal as a monetary

decimal

string repairRecName = (string)(row[0].Value) + ".xml";

/* Populate the Repair Data tab TextFields. */
this.id.Text = (string)(row[0].Value);
this.req_date.Text = (string)(row[1].Value);
this.status.Text = (string)(row[2].Value);
this.email.Text = (string)(row[3].Value);
this.brand.Text = (string)(row[4].Value);
this.model.Text = (string)(row[5].Value);
this.problem.Text = (string)(row[6].Value);

/* Retrieve the remote file for this specific repair id */
rdc.getXMLFile("REPAIR_RECORDS", repairRecName);

/* Populate the fields of the remaining 3 tabs */
populateTxtFields(RepairDataManager.LOCAL_PATHS["REPAIR_RECORDS"]
+ repairRecName);

isUserChg = true; //any modifications past this point are user's

chgdFields = new List<object>();
this.rdc = rdc;
}

/**
 * Sets a few properties of the Parts DataGridView.
 */
private void setupPartsDGV()
{
    this.partsDGV.MultiSelect = false;
    this.partsDGV.ColumnHeaderDefaultCellStyle.Font = new Font(partsDGV.Font, FontStyle.Bold);
}

/**
 * Populate Customer Info, Billing and Shipping tabs' fields.
 *
 * 'repairFile' path to the local copy of the XML data repair file.
 */
private void populateTxtFields(string repairFile)
{
    repairRec = new XmlDocument();
    repairRec.Load(repairFile);
    XmlNodeList children =
    (repairRec.GetElementsByTagName("repair_data"))[0].ChildNodes;

    this.serial.Text = children[0].InnerText;
this.accessories.Text = children[1].InnerText;
this.notes.Text = children[2].InnerText;
this.diagnosis.Text = children[3].InnerText;
this.fix.Text = children[4].InnerText;

children =
(repairRec.GetElementsByTagName("customer"))[0].ChildNodes;
this.first.Text = children[0].InnerText;
this.last.Text = children[1].InnerText;
this.phone.Text = children[2].InnerText;
this.street.Text = children[3].InnerText;
this.city.Text = children[4].InnerText;
this.state.Text = children[5].InnerText;
this.zip.Text = children[6].InnerText;

children =
(repairRec.GetElementsByTagName("billing"))[0].ChildNodes;
this.estimate.Text = children[0].InnerText;
populatePartsDGV(repairRec.GetElementsByTagName("part"));
if (string.Compare(children[2].InnerText, string.Empty) != 0)
{
  this.labor.Text = children[2].InnerText;
}
this.billed.Text = children[3].InnerText;
this.paid.Text = children[4].InnerText;
populateTotal();

children =
(repairRec.GetElementsByTagName("shipping"))[0].ChildNodes;
this.received.Text = children[0].InnerText;
this.returned.Text = children[1].InnerText;
this.carrier.Text = children[2].InnerText;
this.insurance.Text = children[3].InnerText;
this.tracking_num.Text = children[4].InnerText;

/**
 * Iterate over the part entries for a repair and fill
 * the parts table in the Billing tab with a row for
 * each one. Add a part's (quantity * price) to partsTotal.
 */
private void populatePartsDGV(XmlNodeList parts)
{
  foreach(XXmlNode part in parts)
  {
    partsTotal += decimal.Parse(part.Attributes[0].InnerText) *
      decimal.Parse(part.Attributes[1].InnerText, PRICE_STYLE);

    string[] partData = { part.InnerText,
      part.Attributes[0].InnerText,
      part.Attributes[1].InnerText };
    this.partsDGV.Rows.Add(partData);
  }
}
/**
 * Fill in the total cost (parts + labor) field.
 */
private void populateTotal()
{
    if (this.labor.Text != string.Empty)
    {
        this.total.Text = "$" + (partsTotal +
            decimal.Parse(this.labor.Text, PRICE_STYLE)).ToString();
    }
    else
    {
        this.total.Text = "$" + partsTotal.ToString();
    }
}
/**
 * Event handler for TextField value changes.
 */
private void textChanged(object sender, EventArgs e)
{
    if (isUserChg && (!chgdFields.Contains(sender)))
    {
        chgdFields.Add(sender);
    }
}
/**
 * Event handler for "Cancel" button click.
 */
private void cancelBtn_Click(object sender, EventArgs e)
{
    this.Close();
}
/**
 * Event handler for "Save Changes" button click. If changes to
 * any of the repair data fields have been made they are POSTed
 * to the website's data files. If not, the user is informed. It
 * should be noted adding or removing parts from the part table in
 * the 'Billing' tab are saved immediately.
 */
private void saveChgsBtn_Click(object sender, EventArgs e)
{
    /* Initialize the POST data with the action and ID of the repair
       being modified */
    string dataStr = "action=save" + "id=" + this.id.Text;
    object form = null;
    if (chgdFields.Count == 0)
    {
        /* Tell the user no changes have occured */
        MessageBox.Show("No changes have been made!");
    }
    else
    {
for (int i = 0; i < chgdFields.Count; ++i)
{
    dataStr += "&";
    form = chgdFields[i];
    if (chgdFields[i] is TextBox)
    {
        dataStr += ((TextBox)form).Name + "=" + ((TextBox)form).Text;
    }
    else if (chgdFields[i] is RichTextBox)
    {
        dataStr += ((RichTextBox)form).Name + "=" + ((RichTextBox)form).Text;
    }
}

/* POST the changes to the remote script. */
rdc.updateRepair(dataStr);
}

/**
 * Event handler executed when the RepairEditor Form loads.
 * Initializes a tool tip for the "Save Changes" button.
 */
private void RepairEditor_Load(object sender, EventArgs e)
{
    saveChgsTip.SetToolTip(this.saveChgsBtn,
        "Saves changes made to this repair record.");
}

/**
 * Event handler for the "Add Part" button click. Instantiates
 * and shows an AddPartForm.
 */
private void addPartBtn_Click(object sender, EventArgs e)
{
    AddPartForm ap = new AddPartForm();
    ap.OnAddPartOKBtn_Click +=
        new AddPartForm.AddPartOKHandler(addRowPartsDGV);
    ap.Show();
}

/**
 * Event handler that POSTs the data for a new part to a repair
 * ID specified xml repair data file. Updates partsTotal, overall
 * repair total and the parts DataGridView of the Billing tab.
 */
public void addRowPartsDGV(string[] partInfo)
{
    /* Setup POST data */
    string partData = "action=addpart&" +

"id=" + this.id.Text + "&" +
"name=" + partInfo[0] + "&" +
"quantity=" + partInfo[1] + "&" +
"price=" + partInfo[2];

rdc.updateRepair(partData);
this.partsDGVRows.Add(partInfo);

partsTotal += decimal.Parse(partInfo[1]) *
    decimal.Parse(partInfo[2], PRICE_STYLE);
populateTotal();
}

/**
 * Event handler for the "Remove Part" button click. Sends a POST
 * to remove the part from the file associated with a specific repair
 * ID.
 * Updates partsTotal, the overall total and removes the row related
 * to the
 * part being deleted from the parts DataGridView of the Billing tab.
 */
private void remPartBtn_Click(object sender, EventArgs e)
{
    int selectedIndex;
    DataGridViewCellCollection pCells = null;
    if (this.partsDGVRowsSelectedRows.Count > 0)
    {
        selectedIndex = this.partsDGVRowsSelectedRows[0].Index;
        pCells = this.partsDGVRowsSelectedRows[0].Cells;
        rdc.updateRepair("action=rempart&" +
            "id=" + this.id.Text + "&" +
            "index=" + selectedIndex);

        partsTotal -= decimal.Parse((string)pCells[1].Value) *
            decimal.Parse((string)pCells[2].Value,
                PRICE_STYLE);
        populateTotal();
        this.partsDGVRows.RemoveAt(selectedIndex);
    }
}

AddPartForm.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace BAR_Repair_Manager
{
    public partial class AddPartForm : Form
    {
        /* Event declaration. This event is raised when a new part is
         * successfully entered (AddPartForm's TextFields are valid and OK button is
         * clicked). */
        public delegate void AddPartOKHandler(string[] partInfo);
        public event AddPartOKHandler OnAddPartOKBtn_Click;

        public AddPartForm()
        {
            InitializeComponent();
        }

        /* Event handler for the OK button click. Attempts to parse the
         * quantity TextField as an integer and the price TextField as a decimal.
         * Reports an error if either parse is unsuccessful. If successful, raises
         * defined at the beginning of this class propagating the new part's
         * info to the remote repair data file and causing a recalculation of the
         * total cost TextField of the RepairEditor. */
        private void OKBtn_Click(object sender, EventArgs e)
        {
            int q;
            decimal p;
            if (!int.TryParse(this.quantity.Text, out q))
            {
                MessageBox.Show("\" + this.quantity.Text + \\
                "\" is an invalid Quantity!\n" + \\
                "Please enter a valid integer.");
            }
            else if (!Decimal.TryParse(this.price.Text, 
                RepairEditor.PRICE_STYLE, 
                null, out p))
            {
                MessageBox.Show("\" + this.price.Text + \\
                "\" is an invalid Price!\n" + \\
                "Please enter a valid dollar amount.");
            }
            else
            {
                string[] partInfo = { this.name.Text,
                };
            }
        }
    }
}
this.quantity.Text,
this.price.Text

OnAddPartOKBtn_Click(partInfo);
this.Close();

/**
 * Event handler for the "Cancel" button click.
 */
private void cancelBtn_Click(object sender, EventArgs e)
{
    this.Close();
}

BaseURLEditor.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace BAR_Repair_Manager
{
    /***
     * Allows modification of the base address to the repair website.
     */
    public partial class BaseURLEditor : Form
    {
        public BaseURLEditor()
        {
            InitializeComponent();
        }

        /**
        * Overwrite the old remote_base.txt file with the value of
        * the new base address.
        */
        private void OKbtn_Click(object sender, EventArgs e)
        {
            try
            {
                StreamWriter baseURL = new StreamWriter("remote_base.txt");
                baseURL.WriteLine(this.baseURLTbox.Text);
            }
            catch
            {
            }
        }
    }
}
baseURL.Close();
this.Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.ToString());
}
}
}

RepairDataClient.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;
using System.Xml.Linq;
using System.Diagnostics;
namespace BAR_Repair_Manager
{
    /**
     * Communicates with the repair website. Retrieves xml repair data
     * files.  
     * Sends HTML POSTS back to the website's php scripts to reflect user
     * changes
     * made on repair data. Data changes are always propagated to the remote
     * files
     * (i.e. no double data management takes place, local files are always
     * replaced
     * with the remote versions upon saved changes.)
     */
    public class RepairDataClient
    {
        /* Contains the paths and filenames of the website's scripts. */
        private readonly Dictionary<string, string> REMOTE_PATHS =
            new Dictionary<string, string>();

        /* Necessary for HTML POST execution. */
        private WebClient webClient;

        public RepairDataClient()
        {
            webClient = new WebClient();
            parseRemoteBase();
        }

        /**
         * Retrieve a remote xml file. 'pathKey' identifies the remote
         * file.
         */
         ...
* directory contained in 'REMOTE_PATHS' containing the xml repair data file to retrieve.
* /

```java
public void getXMLFile(string pathKey, string repairID)
{
    webClient.DownloadFile(new Uri(REMOTE_PATHS[pathKey] + repairID), RepairDataManager.LOCAL_PATHS[pathKey] + repairID);
}
/**
* Approves or denies a new repair request upon a user's review of the request. Approving a repair moves the repair file to the remote 'active_repairs' directory and Deny removes the request file altogether.
* 'requestID' identifies which repair data file to approve or deny, 'action' approve or deny depending on the user's decision
*/
public void approveOrDenyRequest(string requestID, string action)
{
    /* POST DATA */
    byte[] reqIDBuf = Encoding.ASCII.GetBytes("id=" + requestID + "&action=" + action);

    /* For some reason the POST wouldn't succeed when using the remote address contained in the REMOTE_PATHS. Hence the line below. */
    //postData(reqIDBuf, REMOTE_PATHS["APPROVE_DENY_REQUEST"]);
    postData(reqIDBuf, "http://users.csc.calpoly.edu/~mrshelle/sp/bar/request_repair/" + "active_repairs/approve_or_deny.php");
}
/**
* Saves user's changes to an approved repair. Used by RepairDataEditor.
* 'chgdFields' the repairID along with the (name, value) pairs indicating the data being modified.
*/
public void updateRepair(string chgdFields)
{
    /* POST DATA */
    byte[] chgdFieldsBuf = Encoding.ASCII.GetBytes(chgdFields);

    /* For some reason the POST wouldn't succeed when using the remote address contained in the REMOTE_PATHS. Hence the line below. */
    //postData(chgdFieldsBuf, REMOTE_PATHS["UPDATE_REPAIR"]);
}
postData(chgdFieldsBuf,
"http://users.csc.calpoly.edu/~mrshelle/sp/bar/request_repair/" +
"active_repairs/update_record.php");
}
/**
 * Setup the WebRequest object and send the HTML POST to the remote
 * script.
 * 'data' the data being POSTed.
 * 'remotePath' indicates the remote script to handle the POST
 */
private void postData(byte[] data, string remotePath)
{
    /* C#'s class to handle general POST transmissions. */
    HttpWebRequest WebReq =
        (HttpWebRequest)WebRequest.Create(remotePath);
    /* All remote scripts are set to accept HTML POST data */
    WebReq.Method = "POST";
    /* Our communication is analogous to a HTML form with its
    method=POST
    * and the action set to a script to process the POST variables.
    */
    WebReq.ContentType = "application/x-www-form-urlencoded";
    /* data's length. */
    WebReq.ContentLength = data.Length;
    /* Stream to write POST data too. */
    Stream PostData = WebReq.GetRequestStream();
    /* Write the data to the stream, close the stream. */
    PostData.Write(data, 0, data.Length);
    PostData.Close();
    /* Get the POST's response, would probably be useful to detect
    errors
    * here and report them to the user if any occur. */
    HttpWebResponse webResp = (HttpWebResponse)WebReq.GetResponse();
    /* Print the status code of the POST's response */
    Debug.WriteLine(webResp.StatusCode);
    Debug.WriteLine(webResp.Server);
}
/**
 * Read the remote_base.txt file expected to exist so the program can
 * retrieve
 * remote xml data repair files.
 */
private void parseRemoteBase()
```csharp
{
    string remoteBase = null;

    try
    {
        using (StreamReader sr = new StreamReader("remote_base.txt"))
        {
            string line = null;
            while ((line = sr.ReadLine()) != null)
            {
                remoteBase = line;
            }
        }
    }
    catch (Exception e)
    {
        Debug.WriteLine(e.Message);
    }

    /* Load the remote paths into a dictionary for easy lookup */
    if (remoteBase != null)
    {
        REMOTE_PATHS.Add("NEW_REQUESTS", remoteBase +
                          "new_request/new_requests.xml");
        REMOTE_PATHS.Add("ACTIVE_REPAIRS", remoteBase +
                          "active_repairs/active_repairs.xml");
        REMOTE_PATHS.Add("APPROVE_DENY_REQUEST", remoteBase +
                          "active_repairs/approve_or_deny.php");
        REMOTE_PATHS.Add("REPAIR_RECORDS", remoteBase +
                          "active_repairs/repair_records/");
        REMOTE_PATHS.Add("UPDATE_REPAIR", remoteBase +
                          "active_repairs/update_record.php");
    }
}
```