Web Based Notification System using LAMP Architecture and C++

by

Shadley Davidson

Christopher Lim

Oliver Tran

Final report submitted in partial satisfaction of the requirements for the degree of
Bachelor of Science
in
Computer Engineering
in the
Faculty of Engineering
of the
University of Manitoba

Faculty Supervisor:
Dr. R. D. McLeod

Course Coordinator:
Dr. U.D. Annakkage

Teaching Assistant:
Mr. Paul Card

March, 2007

Abstract

With its growing usage rate and dependency, the World Wide Web can provide a great medium for Amber Alerts. The Web Based Notification system is composed of a database server, web server, and a pop up notification application that interacts with each other to alert a user of an Amber Alert. The web server and database server were designed using the Linux Apache MySQL PHP software stack and the pop up application was written in C++. Missing child information would be stored in the database where police officials can look up and edit information through a collection of web pages that reside on the web server. The pop up application runs in the background of a user's workstation and monitors the web page for updates in missing children information. A pop up is generated, notifying the user of an Amber Alert.

The performance of the Web Based Notification system was successful in notifying the user of an Amber Alert. The client ran smoothly within the task bar of a windows operating system workstation and required minimal resources in terms of CPU time and memory. The web server provided user web server applications for officials to use in looking up and editing missing child information while in a secure environment. The website was composed of embedded PHP scripts into HTML pages. The site interacted with the MySQL database where all the users and missing child data were stored. The entire system was designed to be used by the general public with little or no background in computers and programming.

Contributions

This project is unique in that it provides a feasible low-cost system that takes advantage of todays technologies, including the Internet, to transmit highly important information. With regard to Amber Alerts, there is no equivalent system that uses the Internet as a means of communicating urgent missing children information. At the time of this writing only traditional news broadcasts and a system to transmit wireless text messages to cell phones have been implemented.

The components of this system were divided up amongst the members of the group but critical design decisions were discussed as a team. Consensus was reached on how design should be implemented and the actual implementation was left up to individual members. The major tasks as completed by each author are indicated below:

All Members:

- Theory and Background research, overall design decisions.

Shadley Davidson:

- Programming of the pop-up notification application
- Designing of the update notification system
- Programming of the Graphical User Interface
- Designing the connection to the website from the application

Christopher Lim:

- Installation and configuration of the operating system and database
- Programming of the connection between the database and the web server
- Implementation of the security measures of the system
- Aiding in the programming of the web server scripts

Oliver Tran:

- Graphical designing of the web page
- Administering the functionality of the web page
- Implementation of the login process and registration for users
- Designing secure administration logins for the web page

Acknowledgments

First and foremost we would like to thank our advisor Dr. R. McLeod for his guidance and support throughout the entire process of the project. His feedback greatly aided us in identifying what was necessary in the project implementation and helped us to achieve our goals.

Our thanks goes out to Dr. Udaya Annakkage and Mr. Paul Card for administering the Group Design Project Course and for providing constructive feedback towards our project.

A very special thanks goes out to Michelle Benoit Sergeant #1530 of the Criminal Investigations Bureau (Division #41) of the Winnipeg Police Service who gave us the criteria used for Amber Alerts in Manitoba and for her advice on how to better improve our system.

We would also like to thank Mr. Mount-First Ng from the Tech. Shop who aided us in the purchasing and configuring of the domain name for our website.

Table of Contents

List of Figures

List of Tables

Nomenclature

.NET (Dot Net)	- A framework used to help developers write managed programs (see Managed Code)
Apache	- Web server for most operating systems.
ASP	- Active Server Pages. A specification that enables Web pages to be dynamically created using HTML, scripts, and reusable ActiveX server components.
C/C++	- A well established computer programming language.
C#	- A newer language used in Microsoft's .NET framework.
CFML	- Coldfusion Markup Language. The programming language used to build dynamic web pages that automatically draw their content from a database.
DB2	- IBM's Database 2 software.
DSN	- Data Source Name defined by Microsoft's ODBC.
Garbage Collection	- A heap management strategy where a run-time component takes responsibility for managing the lifetime of the memory used by an object.
GNU	- GNU's Not Unix. Defines Linux operating systems.
HTML	- Hyper Text Markup Languageis the authoring software language used on the Internet's World Wide Web. HTML is used for creating World Wide Web pages.
HTTP	- Hypertext Transfer Protocol is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.
HTTPS	- Hypertext Transfer Protocol Secure. HTTP Over SSL Protocol enabling the secured transmission of Web pages.
IL	- Intermediate Language. A language used to convert different languages into a single one that can be run on a system.
Java	- A well established multi-platform computer programming language with object orientation.
LAMP	- Linux Apache MySQL PHP, software stack used for Linux servers.
Linux	- A Unix-like operating system.
Managed Classes	- A managed class can interact with other managed classes, even if they are written in another language.

Managed Code	- The developer must specify certain aspects of the code, including security and error handling.
Managed Data	- Data can be handled/destroyed by a garbage collector (see Garbage Collection)
MFC	- Microsoft Foundation Classes. A framework used to simplify application creation for the Microsoft Windows Environment.
MySQL	- Open-source database that uses SQL syntax.
ODBC	- Open Database Connectivity Standard. Connection protocol for databases.
PHP	- The PHP Hypertext Preprocessor is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications.
RDBMS	- Relational Database Management System.
Scripting Language	- A specialized programming language used to create scripts that, when inserted into a Web page, control various elements of the page, such as the user interface, styles, and HTML markup.
SSL	- Secure Socket Layer is a commonly-used protocol for managing the security of a message transmission over the Internet.
SQL	- An acronym for Structured Query Language, this is a standard method of conveying information to and from a database.
Ubuntu	- Open source Linux distribution.
Unix	- Operating system developed in the 1960's and 1970's.

Chapter 1 – Introduction

In approximately the last seven years, the number of people who are connected to the Internet has increased by more than 100% [IWS07]. Currently, the percentage of the North American population who are connected is approaching 70% [IWS07, D307, Mad06]. Many people now look to the Internet as a major source of information, and as such, are connected every day. This project hopes to capitalize on the large percentage of Internet users to theoretically design a large scale notification system, focusing on an Amber Alert.

This paper will outline the design and implementation of a generic Web Based Notification System, with an Amber Alert used to create specific requirements. An Amber Alert occurs when a child is kidnapped, and the situation meets a standard agreed upon by law enforcement agencies and broadcasting companies [CFM07]. In this situation, regular television and radio broadcasts are interrupted with the information of the missing child. The main goal of the Amber Alert system is to get the information to as many people as possible. Using the Internet could open up new avenues and result in the immediate notification of many people.

1.1 Background

The criteria used for deciding when to issue an Amber Alert was based on the criteria given by the Winnipeg Police Service and is as follows:

1. The missing person must be under 18 years of age or have a medical disability.
2. After investigation, it is found that the missing person may have been abducted by a stranger or may potentially be in serious bodily harm or death.
3. There must be sufficient information to share with the general public.
4. The abductor can be apprehended or the child can be found within a reasonable time frame.

Once these four criteria have been met, media releases are faxed to the local news media and Amber Alerts are issued during the regular news broadcasts.

Chapter 2 – System Overview

The Web Based Notification System was designed as three separate components::

1. The Database System
2. The Web Server Module
3. The Notification Application (User Interface)

Each one of these has a specific function that can be used independently of each other depending on the final implementation. The basic design can be seen in Fig 1.1.

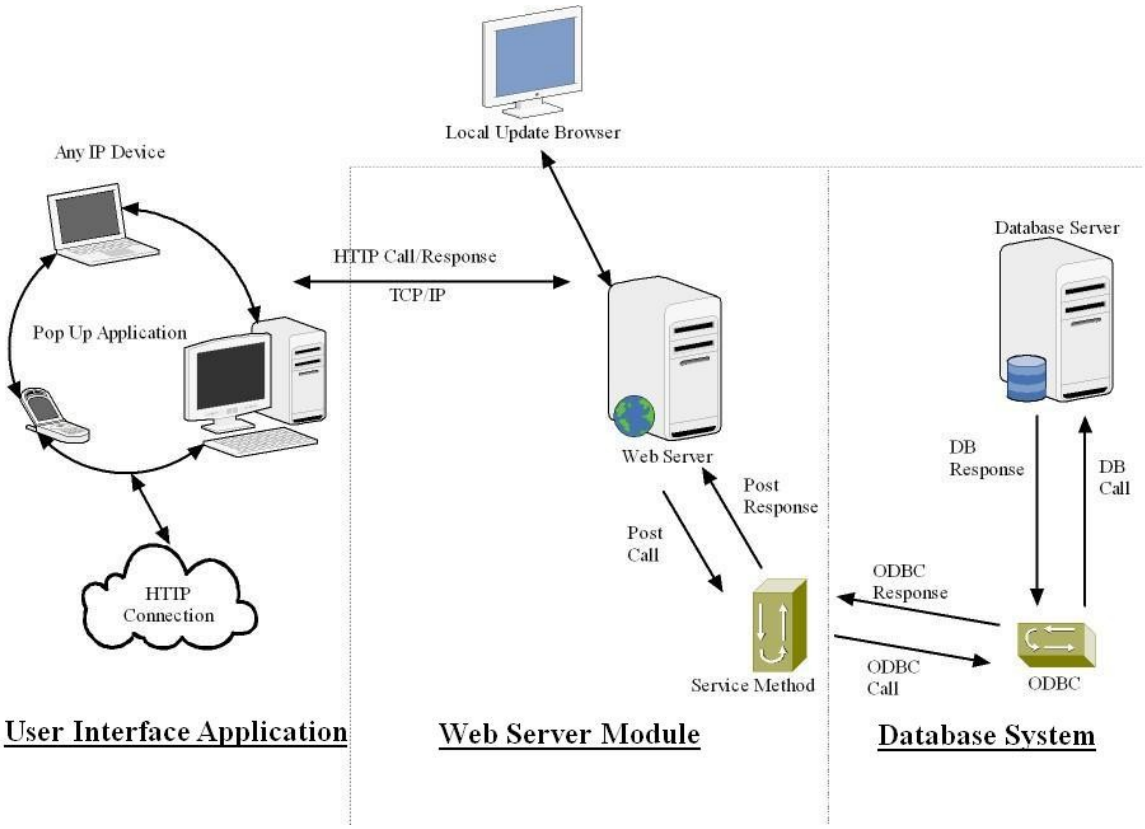


Figure 1.1: The Web Based Notification System

In this implementation, the database houses the information of each missing child. The web server is connected to the database, and will update its pages depending on the information stored in the database. The notification application will connect to the web site, check for missing children, then notify the computer user of any updates. The following chapters will discuss the design and implementation of each of these parts, concluding with an overall system evaluation.

2.1 Components

2.1.1 Database System

The database system comprises of a Ubuntu Linux operating system that runs a MySQL database with installed ODBC (Open Database Connectivity) drivers and connectors. The ODBC is used for MySQL to appropriately communicate with the PHP (Hypertext Preprocessor) scripts that are on the web server. The hardware architecture in which this system is run on is a i586 architecture using Linux kernel 2.6.15-27-386. The hardware specifications include an AMD-K6 3D processor with a clock speed of 500MHz and a 40.0 GB Hard drive running on 184 MB of SD RAM. The database system holds all the necessary information, in our case the missing child data, and provides the information to the web server.

2.1.2 The Web Server Module

The web server is run on the same hardware that the database server is run on and uses the Apache HTTP server software available for the Linux operating system. PHP scripts are used to serve the main web pages and create the login scripts and registration for users. HTML (Hypertext Markup Language) is used in conjunction with PHP to give aesthetically pleasing web sites for users. The web server acts as the medium between the database system and the notification application as it retrieves the information stored in the database and displays it on a web page for the notification application to view.

2.1.3 The Notification Application

The notification application is programmed in C++ and uses a graphical user interface to display messages to the user. The application runs on any workstation running the Windows operating system. The application is updated on a regular basis specified by the user and checks the web site on the web server for updates. New updates trigger a pop-up on the users Windows task bar and gives the option to visit the corresponding web site.

Chapter 3 - The Database System

This chapter will focus on the operating system and database server of the Web Based Notification System. These two components will function as the system back-end that will store the necessary information of the kidnapped child data. The development environment chosen was the LAMP (Linux, Apache, MySQL and PHP) architecture since a "large number of web-sites are built using PHP and MySQL on the Linux platform with Apache as the web server." [Ram05]

3.1 Design Considerations and Alternatives

The design of the database server in the Web Based Notification System involves many design considerations. The following sections will outline some of the choices made by the authors, and discuss the reasons for these choices. The sections that will be discussed are: selection of operating system, choice of database to be used and the methods of connecting to the database.

3.1.1 Selection of operating system

In the software market various operating systems are used to provide users with variety and functionality. The majority of users are most familiar with the Windows operating system since it is the most common operating system used in the software industry. Two options were considered for this project, the Windows operating system and the Linux operating system. Both options were brought up to the following criteria in determining which would best match our project:

3.1.1.1 Operating System Criteria:

1. Performance: Our web server would need to be run on an operating system that would maximize performance and efficiency in order to quickly retrieve and process the necessary data. The data being transferred on this system would need to be transmitted to the users as quickly as possible while reducing time spent processing the data.
2. Reliability: The reliability of the operating system would prove to be valuable in that by not having to spend time maintaining the system, administrators would be able to better spend their time on regular database management. Also, by reducing the

number of times a system needs to be rebooted the probability of losing critical information is greatly reduced.

3. Security: The information contained on our database server, if accessed by the wrong individuals, could result in harm coming to vulnerable persons including children. Being sure that secure connections and access is implemented in our system is an important requirement.
4. Cost: Minimizing the cost of the system makes it more attractive to public sectors that are particularly interested in the transmission of important data such as amber alerts and kidnapped children.

3.1.1.2 Windows XP Operating System

Windows XP "strives for ease of use, with plug and play operation of peripherals and an installation process thats relatively painless" [Dirk04]. While Windows XP is the most common operating system available, it contains a large amount of drivers and software packages that are largely unusable by our Web Based Notification System and would only serve to hinder the performance of our database server. Due to the proprietary nature of the operating system, tweaking the configurations of the operating system is quite limited and therefore reduces the potential of increasing the performance.

Due to the popularity of the Windows operating system constant updates are required to keep the system sustainable and current. This negatively affects our choice of operating system in two ways: The fact that constant security updates are required shows that malicious attacks from users that send viruses and denial of service attacks are quite common with this operating system. This also leaves the systems data prone to vulnerability, having it compromised and potentially stolen by malicious individuals. The second negative result of choosing this operating system is that some of the security and system updates that are done automatically by the operating system require a reboot of the system. As previously mentioned the forced reboots of the system would result in potential down times as well the potential loss of important data.

The Windows XP software is available to most users at a price in the mid $100's to $200's of Canadian dollars. While this cost is minimal, it is still a fairly large cost when compared to other free alternatives.

3.1.1.3 Linux Operating System

Alternatively, the Linux operating system has been known for its achievements in

performance, reliability, and security in the server industry. The Linux operating system is an open-source operating system which means that the source code used to build the operating system is freely available for developers around the world to make changes to it. This provides constant improvement to the operating system and makes for a very dependable operating system.

The Linux operating system offers a wide variety of choice during installation as to what is to be desired in regards to programs being installed. Other than the kernel, any additional features or programs can be included or taken out of the given software package. By making it's operating system completely customizable, it offers the best solution for improving performance on a database server. Through minimizing what is necessary and excluding the software that is not required we can acquire a system with maximized performance. Fortunately the minimizations for determining what is necessary in a database/web server has already been set out in the Ubuntu 6.06 LAMP Server Edition. Ubuntu 6.06 LAMP Server Edition is a distribution of the Linux operating system that provides the minimal amount of software programs required to properly run a database/web server. It also provides the extensibility to add any additional programs that are available for the Linux operating system, making it more customizable if it is discovered later that other programs are required. The Web server portion will be discussed in a later section.

The reliability of the Linux operating system has been well documented and proven to be a sufficiently reliable operating system. It has been designed with the concept that no reboots are required and can provide software updates without the need for a reinitialization of the entire system. This makes the reliability and up time of the system to be quite large and only susceptible to power surges and outages.

The limited usage of the Linux operating system helps in increasing the overall security of the system as most programmers tend to avoid writing viruses that would affect it since they would make minimal impact on society as a whole. The mere conservative construction of the Linux operating system makes it difficult for a virus to appear and cause damage. All of the files on a Linux operating system are strictly owned by its respected owners. A virus would most likely arrive through a users home directory but would find that it could do no damage to the system where the vast majority of the executables are owned and can only be run by the root user.

The fact that the Linux operating system is open-source also makes it difficult for a virus to hide in the program code as the development of the source code is constantly peer-reviewed and being checked on by other experienced programmers.

All these reasons help to prove that the Linux operating system is very dependable when it comes to security concerns.

Being an open-source operating system also means that the operating system is provided to the general public free of charge. This is because the source-code is under the GNU public license and is non-proprietary so no one specifically can claim the source-code as their own since it is a collectively developed entity. The zero-cost as previously mentioned will be particularly attractive to organizations in the public sector to whom would be interested in this Web-Based Notification System.

3.1.2 Choice of Database:

As there are many choices of operating systems there are also many choices of databases. Three options were considered in the choosing of the database system to hold the necessary kidnapped child data: Microsoft SQL Server, DB2 Universal Database and MySQL MySQL was the database that was eventually chosen to hold the kidnapped child data in our Web-Based Notification System.

3.1.2.1 Microsoft SQL Server

Microsoft SQL Server is a relational database management system (RDBMS) found on many industry servers due to it's interoperability with the Windows operating system. Unfortunately as a result, Microsoft SQL Server is only available on the Windows operating system. As mentioned in the previous section we have chosen Linux as our operating system of choice for this project which makes this alternative incompatible with our set up. Using this RDBMS would also have added more cost to our system while there are other alternatives in the market that are open-source and free for use.

3.1.2.2 DB2 Universal Database

DB2 Universal Database is another RDBMS provided by IBM that "delivers a flexible and cost-effective database platform to build robust on demand business applications" (see [IBMDB207]). While traditionally DB2 has only been available to purchase, they have recently released a free version for developers to "download, develop, deploy and distribute"(see [DB2EC07]), which is called DB2 Express-C. It was originally proposed that this database system would be used to test the portability of our system by replicating the database into a DB2 Express-C database and making sure that the functionality of the web-server scripts and application stayed the same. Unfortunately, it was later discovered that the production machine

that we were using to hold our database server did not meet the minimum system requirements in regards to CPU and memory. It was also dropped in favor of MySQL which was already included in the LAMP software bundle that came with the Ubuntu 6.06 LAMP Server Edition.

3.1.2.3 MySQL Database

The MySQL database claims to be the "world's most popular open source database because of its consistent fast performance, high reliability and ease of use. " (see [MySQL07]). MySQL is one of the easiest to use database systems in the market and is provided free of cost since it is an open-source product. The MySQL database was chosen for its simplicity and speed in retrieving information and being able to transmit that information as quickly as possible. The ease of use aspect was particularly attractive since it easily installed and configured itself upon installation of the Ubuntu 6.06 LAMP Server Edition. Another advantage of using the MySQL database was that ODBC connectors are also available on the company's website. These ODBC connectors ended up being our database connector of choice in implementing our design of this product.

3.1.3 Methods of connecting to the database

Two methods of connecting to the database were considered. One was to use the MySQL functions that are already included in the PHP scripting language. Another was to use the ODBC functions that are also available in the PHP scripting language. Their advantages and disadvantages are discussed below:

3.1.3.1 Using MySQL functions

MySQL functions were readily available and used to quickly implement a test system. The functions were easily referenced from the PHP documentation and from other user's experience on online forums. It's ease of use and common use on the Internet proved to make using the MySQL functions a viable choice. Development progressed well with the speed of using the functions and implementing them immediately into the design. The flow chart of the mysql_functions is shown in Fig. 3.1.

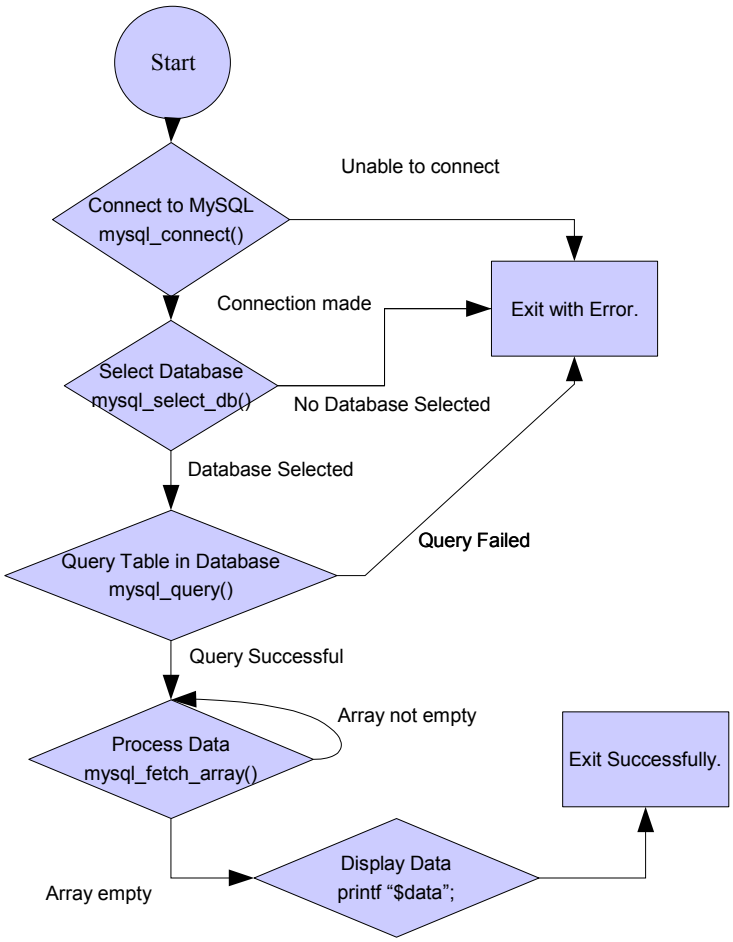


Figure 3.1. Process flow of display.php using MySQL functions.

The disadvantage of using this method was that the functions only work specifically with the MySQL database. This limited the use of our system to that particular database. If the system was to one day be ported to another database, the entire source-code of the web-page scripts would also have to be re-designed and made usable for another particular database. This would amount to tedious programming and added cost. Our supervisor suggested that we try to make our system as portable as possible by using a standard in object database connectivity such as ODBC (Object Database Connectivity Standard).

3.1.3.2 Using ODBC functions

After a basic connection was made using the MySQL functions, the changing of the scripts to use the ODBC functions began. ODBC was a better design choice as this allowed the

scripts for the web site to be portable to any database that has the ODBC connectors installed. Most RDBMS's provide ODBC connectors to interface with their database which gave us more reason to use ODBC as the connection standard. The disadvantage to using the ODBC connectors was that a database could not be selected the same way as the MySQL functions. In MySQL a simple: “mysql_select_db($dbname);” could have been written but this could not be done in ODBC.

A data source name (DSN) needed to be created in the /usr/local/etc/odbc.ini directory for ODBC to perform properly. This data source name needed to be able to use a userid and password that was registered in the MySQL database and be able to know the database name that it is supposed to access. If a different database needed to be accessed another DSN needed to be created. This resulted in more overhead needing to be done for ODBC to work as can be seen in Fig. 3.2.

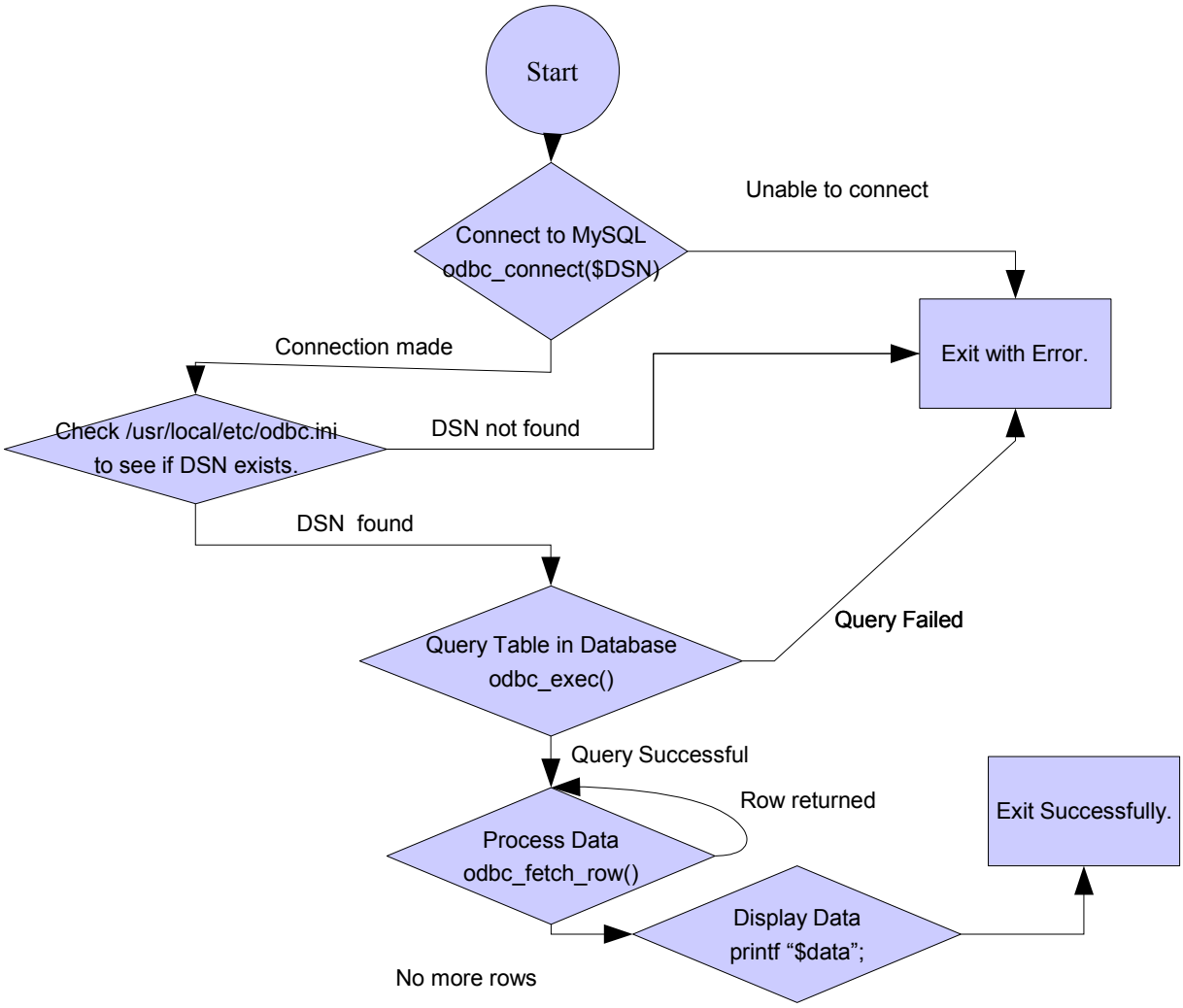


Figure 3.2. Process flow of display.php using ODBC functions.

3.2 Design Implementation

The work required to implement the Web Based Notification System on the database server side includes: PHP scripts to make connections to the local database, access the data and populate the data onto web pages that the web server can service, enforcing security on the system, implementing ODBC to be able to connect to different databases.

3.2.1 PHP scripts design process

3.2.1.1 Original simple script design

In the first stages of our project the only requirement for PHP scripts was to have one script be able to connect to the database, acquire the necessary most up to date information and display the information on to the web page. As stated earlier, this was done using MySQL functions at first and then was changed to use the ODBC functions included in PHP. Once the main script, display.php, was completed, the entire system was checked for functionality and the application was able to acquire the necessary information. When additions to the database needed to be made we accessed the table through the MySQL client on the database server. When we were connected and chose the correct database we were able to add children to the table using SQL (Structured Query Language) syntax. This process was done for the first few months. As children were manually put into the table in the MySQL database it was discussed how usable this would be to administrators who may use this system.

3.2.1.2 Changes to original simple script design

In order to make the scripts extensible to other already existing databases that organizations may already have, it was decided to split up the configuration of the database information into a separate file called config.php. Administrators could customize the configuration file to their own already existing database configuration such as Data Source Name for ODBC (which contains the database name), database user name, database user password, and specific table with the requested kidnapped child information. The opening and closing of the connection to the database were also put in separate files named opendb.php and closedb.php respectively. The regular display.php file was then changed to include the aforementioned files and was used only to display all the contents of the table. The original file display.php displayed the ID, first name, middle initial, last name, a small message, the time and date that the child was added to the database.

As the script developed it was decided that only the latest ten children added to the database should be shown rather than the whole content of the table in order for the application to not have to parse through so much data. Eventually the display.php was changed to remove the message information and only show the ID number, first, middle and last name of the child, City, Province, Country, Priority Level and Date and Time added to the database.

3.2.1.3 Administration script design

After much discussion with the team, it was decided that an easier way for administrators

to access and update the table in the database needed to be implemented. It was agreed that the majority of administrators of this type of system may not have experience with SQL and a simpler method of adding children to the database needed to be developed. An administration script that uses a MySQL user id specified in the /usr/local/etc/odbc.ini file called searcher would be able to add children to the database.

The script created three text boxes that would define the first name, last name, ID number and a small message to be inserted into the database. As the table grew larger and larger it was decided that future administrators would also need to be able to remove children from the database, in the case that there was a mistake or that the child had been found. A "remove" button was created to remove a child from the database by passing the ID number as a parameter to the same script (admin.php). The admin.php file would then check to see if that specific parameter was passed to it and it would delete the corresponding child associated with that ID number. At the output, when the remove button is pressed the page reloads and the corresponding child will be removed from the table displayed.

As the administration page developed, more descriptive information about the child was added to the database and the corresponding administration script needed to be changed to reflect these changes as can be seen in Fig. 3.3. The ID number was changed to be a unique ID and the option to input the ID manually was removed from the admin.php site. This was done so that every child would be assigned a unique ID that would not be modifiable by any administrator.

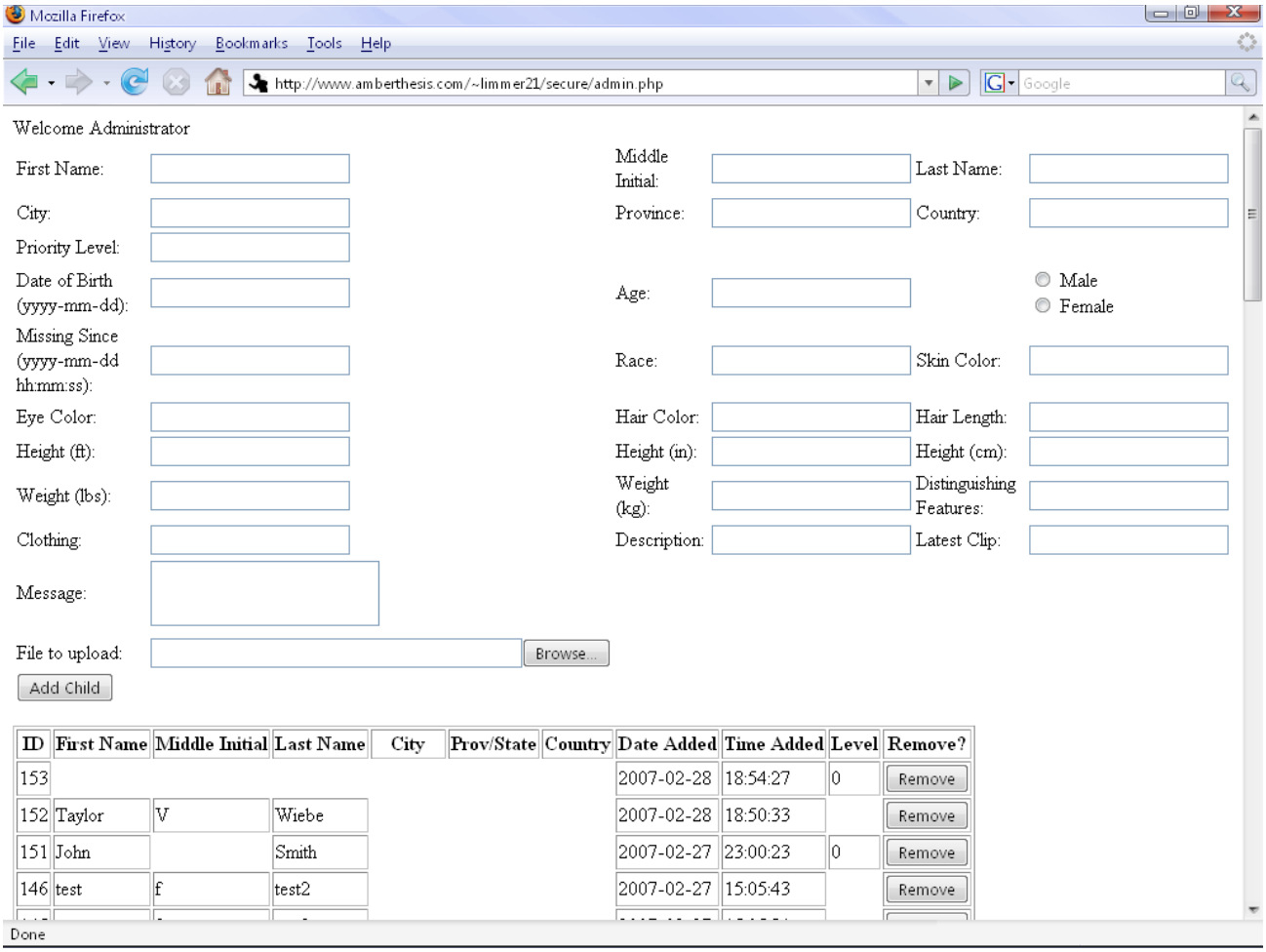


Figure 3.3. Inputting child information into the database for admin.php.

3.2.1.4 Script for application parsing

During development, it was found that the script display.php that was originally chosen to be parsed for data by the application was raising problems during the parsing. It was requested that a simpler script be made to remove the table formatting that was implemented in the display.php file. The simpler script (appropriately named, simple.php) was written to provide the application with an easier to format collection of the exact same data that the display.php presented. The simple.php script presented the data as plain text and the information could be parsed properly by the application.

3.2.2 Security Implementation

With the type of data that is involved in this project it is imperative that the proper security measures are implemented in order to protect the children involved and avoid the

possibility of false data being entered into the database and propagated to the many clients using our program.

When evaluating the types of security that should be implemented we found two areas of concern that needed to be addressed:

1. Only authorized persons and administrators should be able to make changes to the database and table information involved.

2. Login, password and personal information should not be compromised.

3.2.2.1 SSL Implementation

By ensuring that the above two criteria were met, it would guarantee that no nuisances such as advertisement spam or profane messages would be sent over the system. The two criteria would also ensure that falsified data are not entered into the database causing unnecessary alarm. In order to implement the necessary security measures we decided to support data transmission over a secure encrypted connection such as the Secure Socket Layer (SSL). By enabling SSL on our web server we would be able to comfortably use an administrator login script to process the login and password information. The login information and password information would be sent over an encrypted network so that unauthorized users trying to read the data on the network would not be able to. This would not only protect the login and password information but also the registration information on the registration page which allows users to register if they wish to be notified of updates. This would also ensure that only authorized users will have access to change the table information in the database.

3.2.2.2 Application Compatibility Problems Related to SSL

Originally it was proposed that the entire site should be sent over SSL to ease navigation and writing of the scripts. This was thought also to be safer than having information being sent over an unencrypted network. Once SSL was installed it required a certificate to be authenticated on the client side. If the client acknowledges the certificate then data transmission is followed through. While this was good for security it ended up becoming a large obstacle for the application. When SSL is enabled the application is unable to acknowledge the certificate and access the required data. This became a concern as we did not want to have to decide between functionality and security. Further research showed that encrypted data transmission can work with unencrypted data transmission by customizing the web server. The web server was customized as to be able to serve web pages on port 80 (unencrypted) as well as port 443

(encrypted). When this was complete the following diagram (Fig. 3.4) shows the set-up of the system.

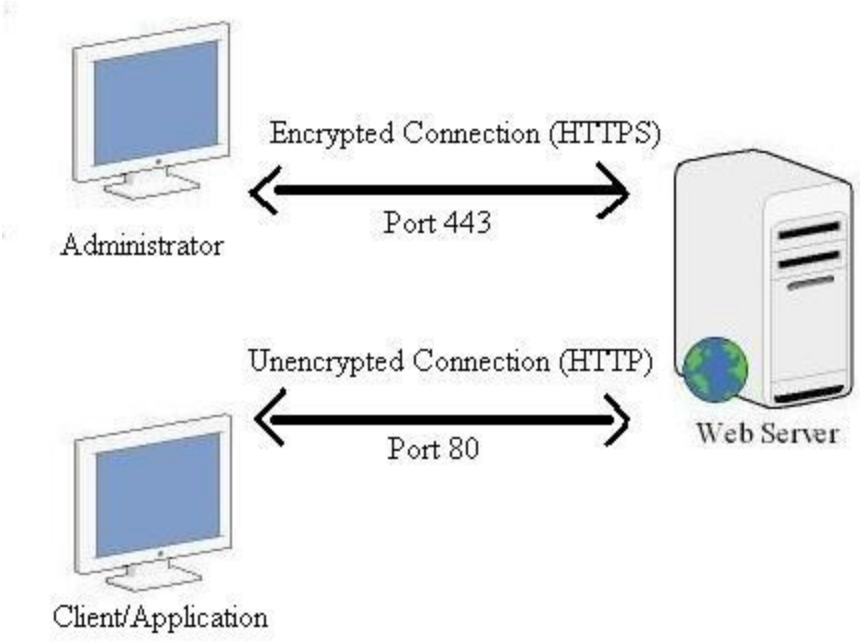


Figure 3.4. SSL enabled Web Server serving data over port 80 (HTTP) and Port 443 (HTTPS) concurrently.

3.2.3 ODBC implementation

ODBC requires two sets of libraries in order to be implemented into our system. The first is the driver manager and the second is the Connector/ODBC driver for MySQL.

3.2.3.1 ODBC Driver Manager

The purpose of the ODBC Driver Manager is to manage communication between the application (in our case the PHP scripts) and the database driver. The ODBC Driver manager "resolves Data Source Names (DSN). The DSN is a configuration string that identifies a given database driver, database, database host and optionally authentication information that enables an ODBC application to connect to a database using a standardized reference" [ODBC07] The Driver Manager is in charge of loading and unloading the specific database driver that is associated with the Data Source being referenced. It is also in charge of processing ODBC calls from the PHP scripts and passing them to the database driver. On Linux operating systems, two

options are available for the ODBC Driver Manager, unixODBC (www.unixodbc.org) and iODBC (www.iodbc.org). unixODBC was chosen to be the Driver Manager for our system due to its helpful documentation and experience with MySQL databases. The latest version of unixODBC available at the time was version 2.2.12. This version was used for our Web-Based Notification System.

3.2.3.2 Connector/ODBC driver for MySQL

The MySQL connector (or ODBC driver for MySQL) is a "library that implements the functions supported by the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL." [ODBC07]. The latest version of the MySQL connector available was version 3.51.12 and this was used at the beginning of the project. The design of the ODBC implementation is shown in Fig. 3.5.

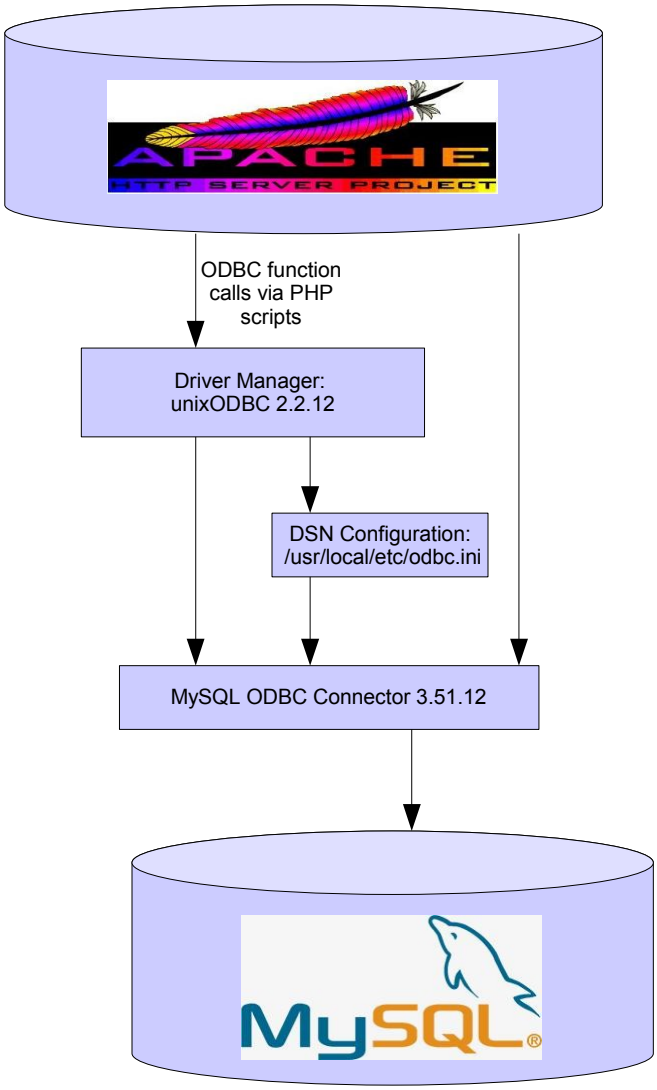


Figure 3.5. Diagram showing relationship between unixODBC Driver manager and MySQL ODBC Connector with the MySQL database and the Apache Web Server.

3.2.3.3 Problem related to ODBC Connector

During development a problem occurred that would cause all the PHP scripts to malfunction with an error warning concerning corrupted char sets. It seemed that this problem occurred directly after the implementation of ODBC. The problem seemed tolerable at first but as more users were connecting to the web server the intensity of the problem increased and created the web server to be greatly unstable. Three modifications to the design of the ODBC implementation were considered:

3.2.3.3.1 Do not implement the ODBC standard.

This option was eventually abandoned due to ODBC being a huge advantage towards the portability of our system. Not implementing the ODBC standard would have forced the administrators of the system to use the MySQL database even if they already had a previous contract with another database system. This option was eventually disregarded as we did not want to constrain the requirements of the system to be so tight.

3.2.3.3.2 Use another ODBC Driver Manager

Another design consideration involved using the iODBC Driver Manager rather than the unixODBC driver manager as it seemed that the problem was specifically with the unixODBC Driver Manager. After further investigation it was found that the problem actually resided with the MySQL ODBC Connector.

3.2.3.3.3 Change the MySQL ODBC Connector version

After the problem had been found to originate from the MySQL ODBC Connector, many measures were taken to try and resolve the problem. They included, upgrading the MySQL database version, recompiling the MySQL ODBC Connector from source code, and trying to apply suggested patches from the MySQL developer bug system.

The solution that was eventually implemented was installing an older version of the MySQL ODBC Connector. Once the older version was installed and tested the problem was resolved and did not re-occur.

Chapter 4 - The Web Server Module

This chapter will focus on the Web Server component of the Web Based Notification System. With the system being composed of a client application sitting on an end users workstation and a remote database server elsewhere, the web server was needed to bridge the gap between these two components. Since this project was aimed for police officials and public use, the web server serves as the main interface between the user and the web notification system. For less computer experienced police officials and the general public, the web server is designed to display simple web pages containing a user friendly environment to interact with the database server.

Figure 4.1 Web Server Flow of Data

Figure 4.1 shows the layout of the web server. In the event of an Amber Alert or voluntarily, a user can visit the web server via any common web browser. The web browser will display dynamically the missing children information directly from the database through the use embedded programs within the web site. The following sections will discuss design criteria of the web server along with alternative design solutions and the web site that resides on the web server along with its scripts and programs.

4.1 Design Considerations and Alternatives

4.1.1 Design Criteria

There are many options to consider when designing the Web Server. The following sections will outline the design choices made when implementing the Web Server. Such sections are: The type of web server to serve public web browsers, the choice of scripting language to interact with the database server, and the layout of the web site. The registered address of the web site is www.amberthesis.com.

4.1.2 Web Server Criteria

4.1.2.1 User Compatibility

A key factor when designing the web server was to provide a user friendly environment for police officials who would have access to the child and administrative databases. To provide a user friendly environment, the web server would have to provide web pages that would contain a graphical user interface (GUI) through any common web browser in order for officials to lookup, modify, and add data into the database without having to log directly into database server and entering Structured Query Language (SQL) commands. With this in mind, the web server will be employing Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) protocols to display web pages to the user to interact with. This would expand the usability to all police officials and the general public who would have no experience in database management.

4.1.2.2 Security

Since the web server is the component that is the most interactive, security will be a critical asset. The database containing important missing children's information must be protected from malicious users. Security would be implemented in the server layer through secure socket layer (SSL), and also in the scripting language layer.

4.1.2.3 Cost Efficiency

The Amber Alert system in Manitoba is run by a partnership between law enforcement agencies and broadcasters and is completely voluntary. Our goal is to have any user to be able to use this system. Because of this, our web based notification system is aimed to be cost effective and will have very low hardware requirements. The web server and its features will be implemented using open source software (freely available) and will run off a Linux operating system to greatly reduce cost and make the project feasible to integrate in to the current Amber Alert system.

4.1.3 Choice of Web Server

Based on their popularity and large support community, there were two main alternatives when selecting the type of web server. The two types of servers to choose from were , APACHE HTTP Server, and Microsoft's Internet Information Server (IIS).

4.1.3.1 APACHE HTTP Server

Apache is an open-source HTTP Web server software. The software license under which software from the Apache Foundation is distributed is a distinctive part of the Apache HTTP Server's history and presence in the open source software environment. The Apache License allows for the distribution of both open and closed source derivations of the source code. It is currently the most popular web server on the Net. It can be run on popular operating systems such as Windows, OSX, and UNIX based operating systems. It is a full featured server with many powerful add-ons freely available. Apache is primarily used to serve both static and dynamic content on the World Wide Web. Many web applications are designed expecting the environment and features that Apache provides.

4.1.3.2 Microsoft Internet Information Server

An alternative to Apache is Microsoft's Internet Information Server (IIS). This web server is Apaches main competitor and is also widely recognized and supported and has similar feature as Apache. It is the world's second most popular web server in terms of overall websites. IIS runs only on Microsoft's Windows operating system.

With these two choices for web servers, APACHE was agreed to be the better choice. It was chosen mainly because of its open source nature and can be run on Linux operating systems. This would remove the need for cost for licensing other competing operating systems. The popularity and greatly vast support for APACHE also gives it a slight advantage over Microsoft IIS.

4.1.4 Choice of Scripting Language

A scripting language is required to produce a program that is to display information dynamically from the database server onto the web sites. These programs are appropriate whenever you a page is to be created dynamically when the browser requests the page. An example would be to display date, time, and other information in different ways. Often, these program scripts will query a database and format the data retrieved for display in a HTML page. The programs written in these languages would be embedded into the current Hyper Text Markup Language (HTML) script used to display the web pages. Although there are many languages to be chosen from, there were three that were focused on because of their popularity and large support community. The three choices for scripting languages were Hypertext Preprocessor (PHP), Macromedia ColdFusion Markup Language (CFML), and Microsoft Active Server Pages

(ASP.NET).

4.1.4.1 Hypertext Preprocessor (PHP)

PHP (recursive acronym for PHP: Hypertext Preprocessor) is an open-source server-side scripting language for creating dynamic Web pages for e-commerce and other Web applications. A dynamic Web page is a page that interacts with the user, so that each user visiting the page sees customized information. Dynamic Web applications are common in Web sites, where the content displayed is generated from information accessed in a database or other external source. PHP offers a simple and universal solution for dynamic Web pages. The intuitive interface allows programmers to embed PHP commands right in the HTML page. PHP's syntax is similar to that of C and Perl, making it easier to learn for a person with previous C or Perl experience. Its elegant design makes PHP significantly easier to maintain and update than comparable scripts in other languages.

4.1.4.2 Macromedia ColdFusion Markup Language (CMFL)

CFML is a Web page markup language that allows a Web site developer to create pages with variable information including text and graphics that is filled in dynamically in response to variables such as user input. Along with the usual HTML tags that determine page layout and appearance, the page creator uses CFML tags to bring in content based on the results of a database query or user input. CFML tags perform all server-side tasks such as database queries by condensing complex processes, that would normally require knowledge of programming languages such as Java or C++, CFML has gotten praise for its ease of use because with its integration with Macromedia's Dreamweaver software.

4.1.4.3 Microsoft Active Server Pages (ASP.NET)

ASP is a server side programming language that enables the creation of dynamic and interactive web pages that are not affected by the type of browser the web site visitor is using. ASP pages are scripts that are run, or executed, on the web server. The script is interpreted from top to bottom to create HTML pages that are sent to the browser for display. Because ASP is a Microsoft product, it was designed and optimized for IIS web servers.

Although it was the most difficult language to work with, the final decision was to go with PHP scripting. This was mainly due to its open source nature and packaging with APACHE

in the LAMP (Linux Apache MySQL PHP) bundle. ColdFusion was enterprise level software and would require a high cost to license. ASP ran optimally on Microsoft Windows platforms as opposed to PHP's versatility on any platform. This versatility gave PHP a slight advantage.

4.1.5 Web Site Design

The website serves as the users graphical user interface to view information stored on the remote database. For administrators, the web site is also the means to edit, add and post data through simple straight forward forms and buttons. The original website was aimed to be two pages that users will see: the main page, which would display missing child information in a table format, and administrative page that only officials would have access to in order to add and update the child database. More features and pages were added to the prototype design to demonstrate features and flexibility of the web site. The flexibility allows the Amber Alert authorities to redesign the site as they see fit.

4.1.5.1 User Registration

A registration form was added to store user information within a separate database designed to store user information. These attributes can be called later for filters for data display. Such attributes that would be used to display specific information to a user would be updates regarding city, or location of the user and email of the user to notify them of updates to the client program and site. Each user can have a password of their choosing where it would be encrypted and stored in the user database.

4.1.5.2 Login System

Although the information displayed is meant for the general public, a login system is required for personnel with authority to access the database securely in order to modify it. In other applications other than an amber alert, the user login system can be used for general users as well if the design called for only registered users to be able to view database information.

4.1.5.3 Embedded PHP and HTML

Each page is composed of HTML scripting code along with some embedded PHP. Because the web site is made up of more than one page, a form of passing user variables from one page to another is needed to perform certain functions.

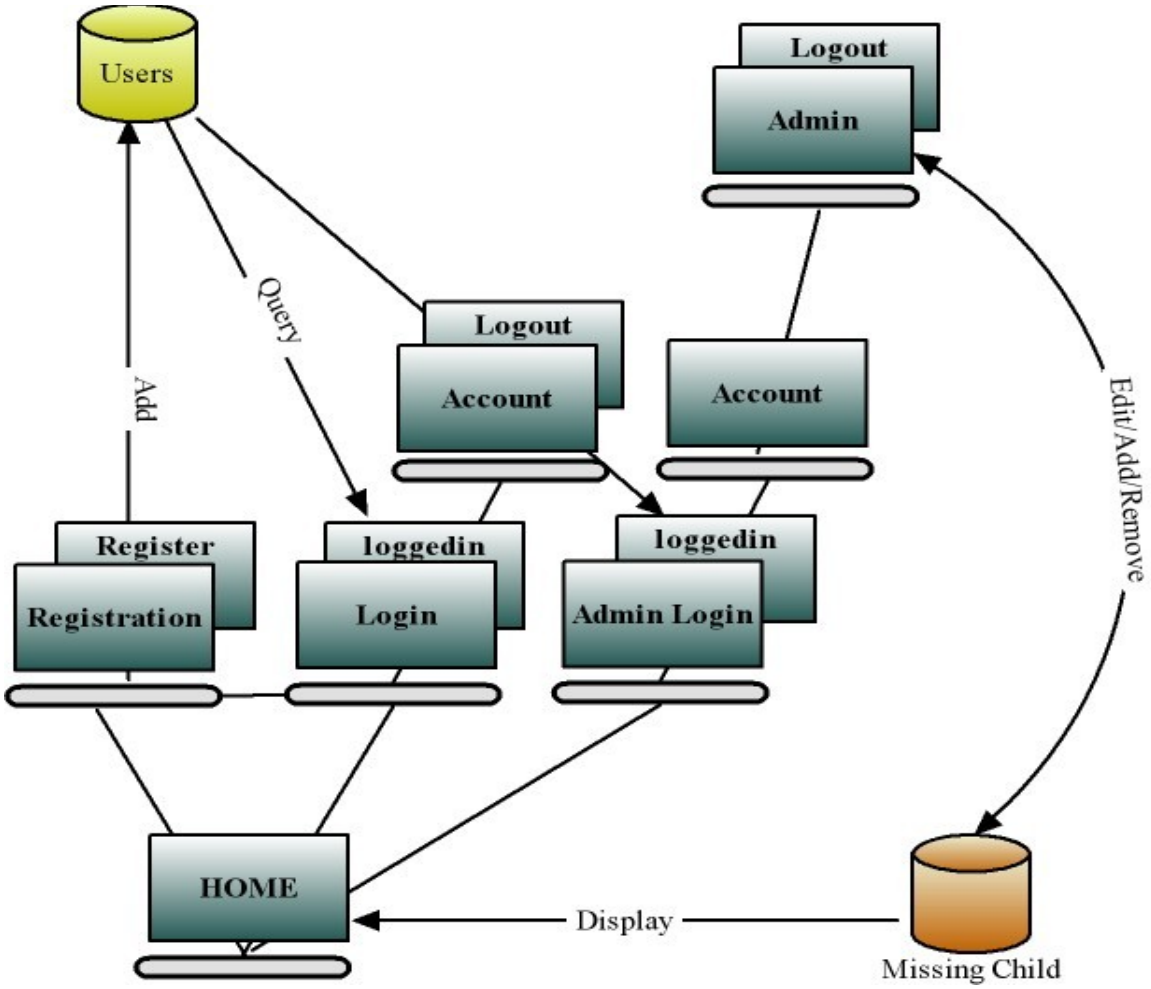


Figure 4.2 Layout of the Web Site

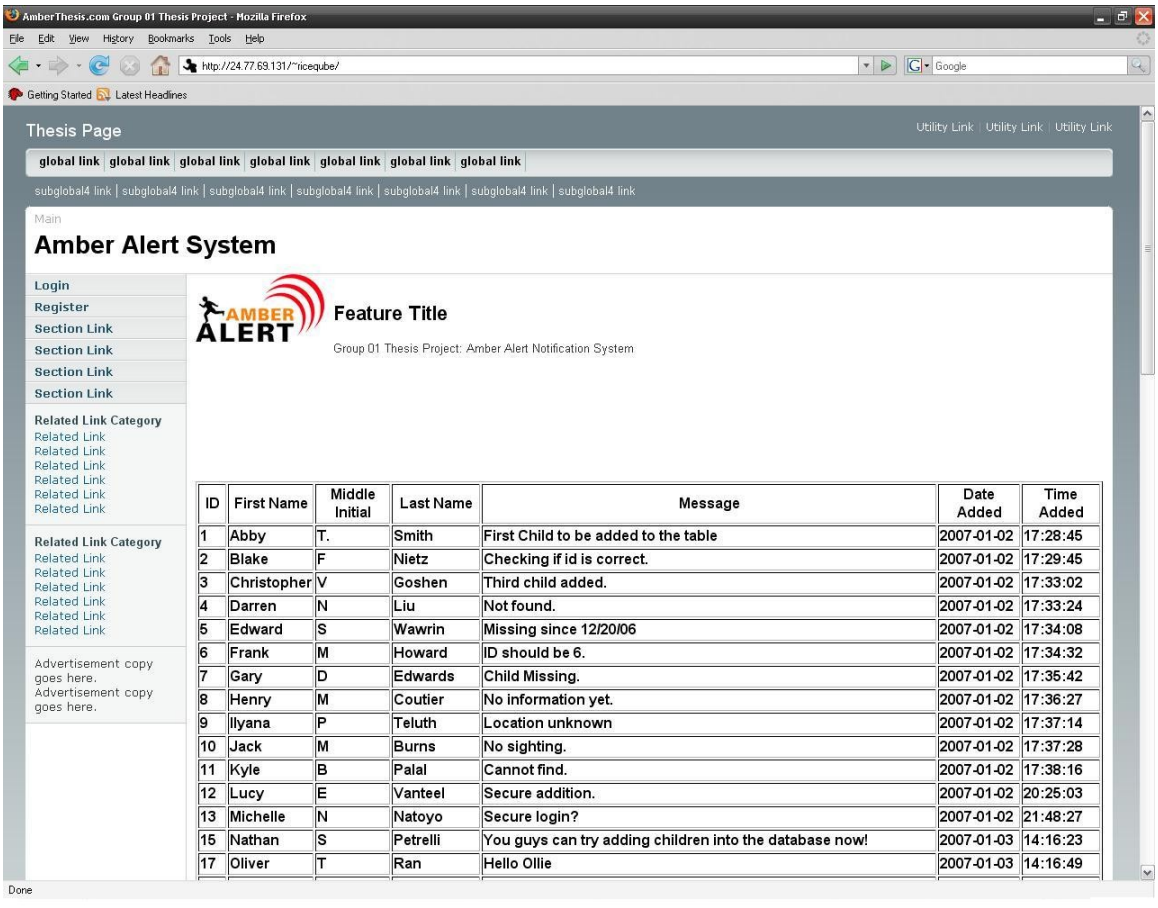
Figure 4.2 displays the design layout of the web site. There are two databases: MISSING CHILD database and the USERS database. The "HOME" page will be the first page that users will see when they visit the site. The home page will have direct reference to the database and display missing child information to anyone who is interested. Also from the home page, users have access to links to the user registration and login pages. Each block represents a script containing HTML and PHP programs. Blocks in the front represent pages that a user will see on their web browser. Blocks sitting behind represent support scripts that are executed "in the background" and will direct to a display script page when done executing. The only other page that allows access to the missing child information is the administrator's page. This can be accessed through a separate login. The log in and registration modules are used to modify and access the users database. Depending on the type of login, a different table would be selected (admin or public). Registration is not required for administrators because a select few are only allowed access to the child database. To allow another, the web master would manually add an

admin to the database. A script can be written for registering officials but will not be linked to the home page or can be accessed by the public for security reasons.

4.2 Design Implementation

4.2.1 Web Pages

This section will discuss the main web pages that make up the Amber Alert web site. The key pages that make up these sites are index.php, user_registration.php, and login_system.php. Along with these pages, a number of support scripts that these pages refer to and are needed to function. All HTML coding was written using Macromedia Dreamweaver whereas certain PHP scripts where also written in Dreamweaver along with basic notepad.



ID	First Name	Middle Initial	Last Name	Message	Date Added	Time Added
1	Abby	T.	Smith	First Child to be added to the table	2007-01-02	17:28:45
2	Blake	F	Nietz	Checking if id is correct.	2007-01-02	17:29:45
3	Christopher	V	Goshen	Third child added.	2007-01-02	17:33:02
4	Darren	N	Liu	Not found.	2007-01-02	17:33:24
5	Edward	S	Wawrin	Missing since 12/20/06	2007-01-02	17:34:08
6	Frank	M	Howard	ID should be 6.	2007-01-02	17:34:32
7	Gary	D	Edwards	Child Missing.	2007-01-02	17:35:42
8	Henry	M	Coutier	No information yet.	2007-01-02	17:36:27
9	Ilyana	P	Teluth	Location unknown	2007-01-02	17:37:14
10	Jack	M	Burns	No sighting.	2007-01-02	17:37:28
11	Kyle	B	Palal	Cannot find.	2007-01-02	17:38:16
12	Lucy	E	Vanteel	Secure addition.	2007-01-02	20:25:03
13	Michelle	N	Natoyo	Secure login?	2007-01-02	21:48:27
15	Nathan	S	Petrelli	You guys can try adding children into the database now!	2007-01-03	14:16:23
17	Oliver	T	Ran	Hello Ollie	2007-01-03	14:16:49

Figure 4.3: index.php displayed on a Web Browser

4.2.1.1. Index.php

Index.php serves as the main home page that the user will see upon visiting the site. HTTP requests made from the browser upon reaching a site traditionally searches for a web page with the name "index", hence the home pages name. The home page is designed with a minimalist approach where its main functionality is to display the missing child information and provide links to further navigate. Figure 4.3 shows the layout of the home page. It has a traditional homepage look with navigational links on the left hand side along with ample space reserved for web advertising if desired. The structure of the page is coded using a series of HTML beginning and ending tags. In the center of the page, the missing child information is displayed in the form of a table. The table is displayed by embedding a PHP script in the section of the HTML code where the table is to be displayed.

4.2.1.2 User_Registration.php

This script is designed to take in a general user's information when registering on to the website. For the prototype version of the web site, the registration form does not grant a user special access or privileges if a user decides to register. If desired by authorities, the purpose of the user registration script can be used to keep track of users and grant them access to download the pop-up client software. This script is mainly composed of HTML with some key PHP sections.

The code starts with a PHP script that is used for disabling caching to prevent stored information in the browsers temporary files and to preserve up to date information when data is sent to and from the browser. This is achieved by the "*header("Cache-Control: no-cache, must-revalidate")"* line for HTTP version 1.1 and the *"header("Pragma: no-cache")"* line for HTTP version 1.0. A session is then started for the user. The amount of data to be loaded was minimal so disabling caching did not effect the time it took to load up the web page.

A session-enabled page allocates unique identifiers to users the first time they access the page, and then re-associates them with the previously allocated ones when they return to the page. Any global variables that were associated with the session will then become available to the code. These variables are stored within the session array $_SESSION[key], where key can be any type of index for the array (integer, or text). Cookies are an alternative to sessions. Cookies are files that are stored onto the clients workstation and contain information to be referenced if the user returns to the site. The reason sessions were decided to be implemented over cookies was to

avoid security measures with operating system security and virus scanners. The minimal amount of information to be stored was not significant enough to require a time saving approach such as cookies.

An immediate action performed once the session is started, is a check whether the session has been initiated before. If it hasn't, the script will include another PHP script called session_init.php in which its role is to simply define some specific variables within the session array. Such variables include the database name, what table to look up, the login and password to connect to the database.

The second task is to check a flag status to see if there were any exceptions that has been generated. If so, then the corresponding message would be displayed at the top of the screen. After submitting user information, there is a chance where the data would not be in the correct format or not exist at all. Such exceptions would include a null entry within a field and entries that are too long to be entered. The exception would be caught, the flag set to the appropriate value, and the page refreshed with the new flag status where the error would be displayed.

The remainder of the code is now for displaying the form for which the user fills out their information to be submitted. The type of fields varies from a standard text box to a pull down menu to demonstrate the customization of a user registration form that an amber alert system or any other type of implementation would require. These forms are then mapped to PHP variables and stored within the session array. For the prototype script in this case, a "reset" button and a "fill with samples" button were added to clear all data and quickly fill in the forms to speed up testing of the page (shown in the figure below). The majority of the code in this section describes how the form is to be displayed and is mainly done in HTML.

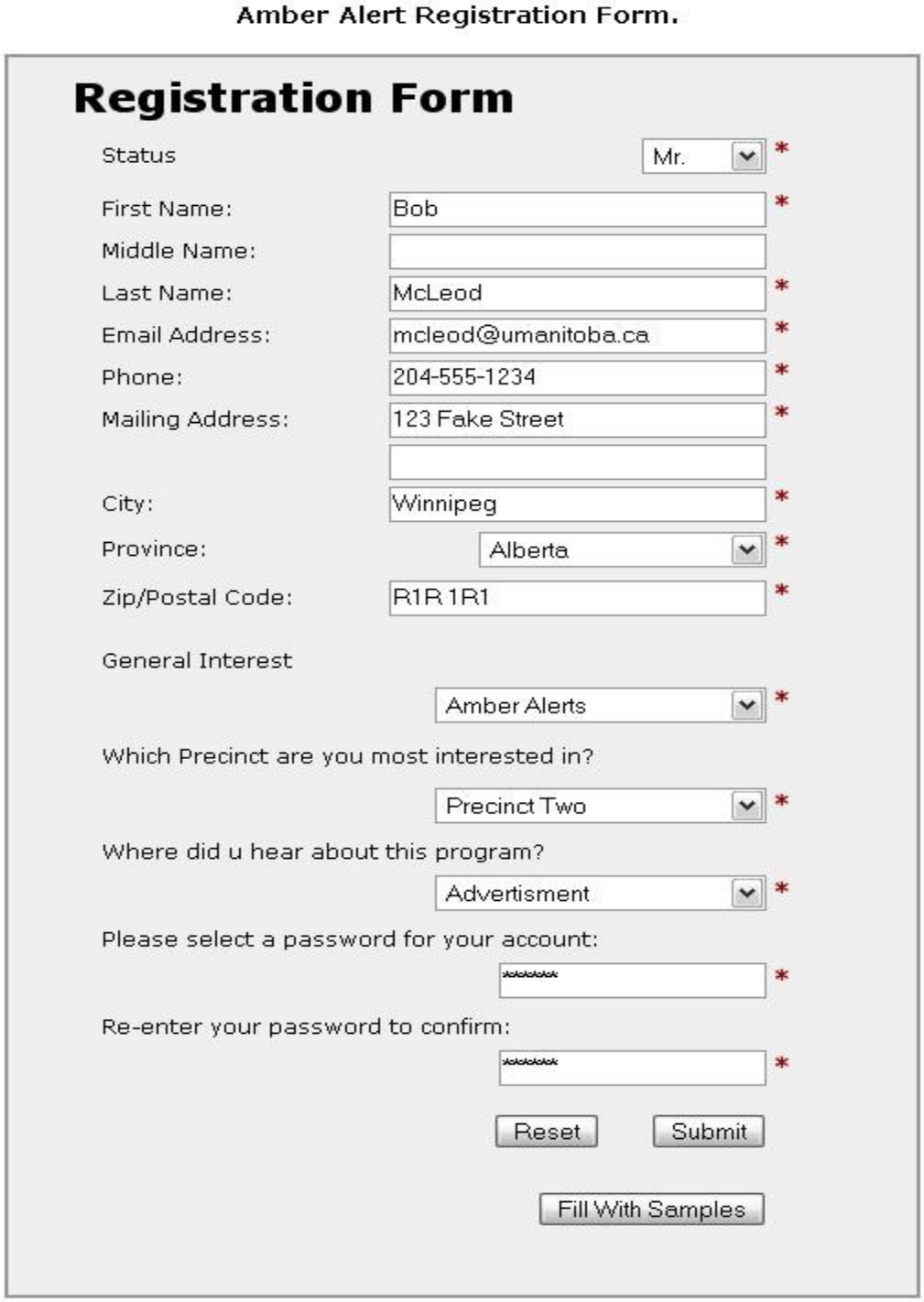


Figure 4.4 Screen shot of the registration form

Figure 4.4 shows what the browser rendering of user_registration.php. The form questions are basic questions and can be later on used as filters for other types of alerts other than amber alerts if further implemented. Upon hitting the submit button, the user input is sent to a

script named register.php.

4.2.1.3 Register.php

This script takes in the parameters sent from user_registration.php and retrieves them from the session array. It checks the format of each array item and makes sure that they are in the correct format. If not, the flag status is set and the script redirects the user back to the registration screen with the new corresponding error message.

Its next task, providing the error flag was not set, is to check for duplicates in the user database. This is done by checking though users email addresses. All the user parameters are copied to a local array and the email address string is used to query the user database. This is achieved using the Structured Query Language (SQL) command "COUNT". The query string *"SELECT COUNT(sEmail) FROM tbl_users where sEmail = '".$_SESSION['email']."'"*, is then saved to a variable and used as a parameter in an ODBC execute command. This command queries the user database and generates a count of instances of the given email address string. The result is returned and stored in a variable where it is checked to see if the number is more than one. If the email address exits then, the flag is set again, and redirected back to the user registration page occurs along with the proper message.

Before inserting to the database, a key operation is done involving the user's password. Within the user database, the password field is set up to store a character string of up to sixty characters. The reason for the large string is to accept an encrypted form of the password. The user's password is encrypted using Message Digest 5 (md5) encryption. Message Digest 5 is a standard algorithm that takes as input a message of arbitrary length and produces a 128-bit fingerprint or message digest of the input. Any modifications made to the message in transit can then be detected by recalculating the digest. This provides a one way hashing of the password where it is stored in the database in its encrypted form. In the case of a malicious user getting access to the user database, they would still have to decrypt the password.

Finally all user variables, including the md5 encrypted user password , are concatenated into the following string:

```
$query = "INSERT INTO tbl_users (sFName, sMName, sLName, sAddr1, sAddr2, sCity, sState,
sPCode, "."cCountryCode, sPhone, sEmail, sPassword, sQ1, sQ2, sQ3, sAccessPeriod)
"."VALUES (".$arVals['fname']."., ".$arVals['mname']."., ".$arVals['lname'].".",
```

".$arVals['addr1']."', '".$arVals['addr2']."', '".$arVals['city']."', '".$arVals['state']."',
'".$arVals['pcode']."', 'US'".", '".$arVals['phone']."', '".$arVals['email']."', '".$arVals['passwd']."',
'".$arVals['q1']."', '".$arVals['q2']."', '".$arVals['q3']."', '".$arVals['access_period']."')";

The string variable $query is then used as a parameter for the ODBC execute command. The user's attributes are then stored into the user database and is assigned a user id number. The user id number is an arbitrary number assigned to registered users and is used for logging in.

4.2.1.4 login_system.php

Figure 4.5 Login Screen

Figure 4.5 shows the login screen that the user will see when logging into the system. The first check this script does is check a key Boolean value called loggedin. This value is stored within the session array and will be explained in the next section. If loggedin is true, the user is automatically redirected to a page called account.php which will later be discussed. If false, the script resumes normally. This script works similarly to the user registration script but much simpler. In this case, this script only accepts two inputs from the user. The user's user id number and password is required in order to login. Caching is disabled, a session is started, error flags are checked for and the user input is stored within the session array upon hitting the submit button. The data is then sent to a script named loggedin.php

4.2.1.5 loggedin.php

The functionality of the login system simply involves a Boolean value named ‘loggedin’. Upon the calling of loggedin.php, ‘loggedin’ is set to false automatically. The script then looks for a single exception where null is entered for both fields from the login screen and generates a flag status and redirects back to the login screen if necessary.

The script then takes the user id number and creates a query string for ODBC to execute. The query asked, returns all the information about the user that was stored. Another check for an exception is made where the user name does not exist and a flag/redirection to login is generated if needed.

Since md5 encryption is one way only, the password submitted from the login screen is also encrypted and then a string compare is made between the encrypted password from the database and the given password in its encrypted form. Upon a match, then and only then is loggedin made true. As long as the user remains on the website and the browser is remained open, the user will be granted access to secure pages that require a user to be logged in to be viewed.

Secure pages will always begin by starting the session up and checking to see if the loggedin value is set to true. If loggedin is at its default value of false, the user is automatically redirected to the main home page. If loggedin is true, then the script resumes, and the user is allowed to browse through the secured page. For the prototype page, upon logging in, users will be sent to account.php.

4.2.1.6 Account.php

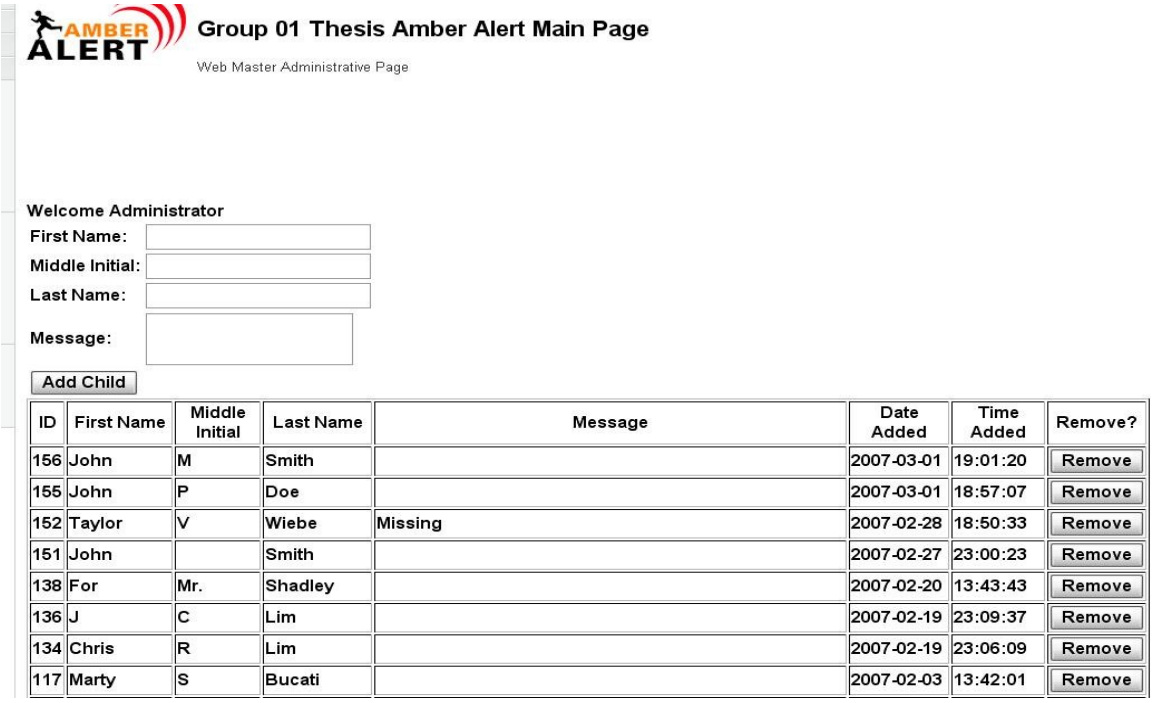
This page is the only secure page to the public user. It was designed to demonstrate how a user must be logged in to view this page. The script simply checks the value of loggedin within the session array and determines whether or not to redirect the user or not. This page also displays the users information retrieved from the database upon logging in for demonstration of the session array system. From this page, the user’s only option is to log out through the logout.php script.

4.2.1.7 logoff.php

This script is a simple script designed to log out the user from the current session they are in while navigating through the website. This scripts only function is to destroy the current session. This is achieved by calling up the session, unsetting the session, and then destroying the session through PHP commands. The user is then redirected back to the home page.

4.2.1.8 Web Master Scripts

Web master scripts are duplicate versions of the registration, login, and account page with an additional admin page. These scripts are used to register a new official with granted access to the user and missing child database, log in the new official, and grant access to the add/remove/edit script for the child information. The way these scripts function is the same as the earlier mentioned scripts except all user information is stored in a separate table within the user database. This table is different then that of the public user table. Upon logging in, the webmaster account page grants the official a link to the admin page where the official can add, edit and remove a missing child.



ID	First Name	Middle Initial	Last Name	Message	Date Added	Time Added	Remove?
156	John	M	Smith		2007-03-01	19:01:20	Remove
155	John	P	Doe		2007-03-01	18:57:07	Remove
152	Taylor	V	Wiebe	Missing	2007-02-28	18:50:33	Remove
151	John		Smith		2007-02-27	23:00:23	Remove
138	For	Mr.	Shadley		2007-02-20	13:43:43	Remove
136	J	C	Lim		2007-02-19	23:09:37	Remove
134	Chris	R	Lim		2007-02-19	23:06:09	Remove
117	Marty	S	Bucati		2007-02-03	13:42:01	Remove

Figure 4.6 sAdmin.php

Figure 4.6 shows sAdmin.php. This page allows an official to edit missing child information through the ease of a simple form. This removes the need for an official to login to

the SQL database and write lengthy syntax that would require a steep learning curve.

4.3 Observations

When implementing user forms to be parsed and uploaded into the database, the threat of SQL injections surfaced. SQL injection is a technique used to take advantage of non validated input vulnerabilities to pass SQL commands through Web applications for execution by a supported database. Malicious users take advantage of the fact that programmers often chain together SQL commands with user provided parameters, and can take this opportunity to embed SQL commands inside these parameters. The result is that the attacker can execute arbitrary SQL queries and commands on the database server through the Web application. An example would be where a malicious user where to login by entering “a' or 't'='t” for a user id. Upon submission, and parsing the input, the result query for the database would look like:

SELECT * FROM users WHERE name = 'a' or 't'='t';

If this code were to be used in an authentication procedure then this example could be used to force the selection of a valid username because the evaluation of 't'='t' is always true.

To prevent SQL injections, all scripts involving forms perform a PHP command called ‘magic quotes’. What this command does is takes in the input strings and adds backslashes to all single quotes, double quotes, backslashes, and null characters. The added backslash “escapes” these characters and prevents a Boolean argument. Although not the best way to defend against SQL injections, magic quotes was a simple approach to solving this problem and did not require much research into the subject which would diverge from the main goal of the project.

Chapter 5 - The Notification Application

This chapter will focus on the end-user notification application of the Web Based Notification System. It will function as the main method for notification to the user. This was designed as a dialog-based Windows application using C/C++ with the Microsoft Foundation Classes (MFC) library. The design environment was Microsoft Visual Studio 2006.

5.1 Design Considerations and Alternatives

The design of the notification application in the Web Based Notification System involves many considerations. The following sections will outline some of the choices made by the authors, and discuss the reasons for these choices. The sections that will be discussed are: the alert cue design, software design choices, and the update notification design.

5.1.1 Alert Cue Design

5.1.1.1 Interface Design

Since the main goal of this project was to design a system that would be used by many different people of varying technical computer skills, it was important that the interface was easy to use for beginners, but also something that computer users would find familiar. Since the actual implementation is dependent other factors, it will be discussed in detail in a later section.

However, the basic desired layout can be seen in Fig. 5.1.

Fig 5.1: Ideal Application Interface

5.1.1.2 Alert Notification

In the event of an alert, a user would need to be notified. There are a few well know ways of doing this in a computer environment, the two most popular being :

5.1.1.2.1 Auditory

A user could be notified using a specific sound. The main advantage of this method is that a user can be away from the computer but will still be able to be notified. A sound also means that current work on the computer does not get disrupted as with a visual notification.

5.1.1.2.2 Visual

A user could be notified using a visual cue, such as a pop-up window. The main advantage of this method is that there is virtually no way to miss a visual cue. If computer speakers are off, or there are none, which can be the case in many work environments, an audio cue could be missed. The main disadvantage of this system is that it could be a disruption to work.

It was decided that the notification would be a visual cue, in the form of a small pop-up window. The goal would be to create a notification in the same form as popular Microsoft applications Outlook, and MSN Messenger. However, an emphasis should be placed on making the notifications as unobtrusive as possible.

5.1.2 Software Design Choices

In the current state of software design, there are many computer languages to choose from, each having certain advantages and disadvantages. There were three languages that were considered for the current application: Java, C/C++, and C#/J#/.Net.

5.1.2.1 Java

Java is a well established computer language developed by Sun Microsystems. Its main advantage is that code can be written on one platform, such as Windows, and work on multiple others, such as Mac, and Linux computers [Sun07]. This would allow the notification application to be used by potentially the greatest number of users. In addition, the authors of this paper have had the most experience in their university education with Java, allowing more of a focus on design, rather than computer coding problems. The main disadvantage of Java is that for the application to be run, the Java Virtual Machine needs to be running as well. Any applications in Java are translated by this virtual machine into code that the host computer can understand. This allows the platform independence, but also slightly reduces speed. Another consideration is that since it was decided that the application will produce a pop-up window, security issues on the

host computer will need to be taken into account. For example, in a Windows XP environment, the pop-up window will need to take control of the screen to display its message, no matter what other applications are currently running. If Java was to be used, the virtual machine would need to take control. While this may in fact be possible, the authors decided that since the majority of the users would be using Windows based PCs, that it would be more safer and more reliable to use a Microsoft based framework.

5.1.2.2 C/C++

C and C++ are two very popular, interconnected languages. C++ is an extension of C with added functionality similar to Java in the form of classes, operator overloading, and exception handling [Wik07]. Having a class structure allows portions of the code to be reused in other projects. It also allows entire sections of the project to be changed without having to alter the way the rest of the project functions. Operator overloading allows a designer to create classes that enhance functionality of already well defined frameworks. Exception handling allows developers to create their applications with a known method of resolving errors resulting in the greatest system stability. A disadvantage of using C/C++ is that the compiler used to translate the source code into an application will result in an application that cannot be used on multiple platforms, only the one that it was compiled for. This will reduce the potential user base for the application, but using a known Microsoft compiler will allow for use of the Microsoft Windows framework. The authors also have significant experience using the C language, which is a good base for the C++ language.

5.1.2.3 C#/J#/.Net

C# and J# are two relatively newer languages that are used within the .NET (dot net) framework. This framework was designed by Microsoft to allow developers to create completely system safe code, meaning that security and error handling are explicitly defined and implemented by the operating system. C# is a language very similar to C++, and J# is a language very similar to Java. Both of these languages can be used to create code that is converted to the .Net language and run on Windows computers. The major advantage of using the .NET framework is the reliability of the project. The major disadvantage is that the authors have never used C# or J# and are unfamiliar with the proper programming practices for writing managed code. This potential learning curve would detract from the emphasis on the design of the project and result in misused time.

It was decided that the application would be written in C++ using Microsoft Visual Studio as the design environment/compiler. This would allow for the greatest number of benefits with the fewest trade-offs:

1. Authors are familiar with C, a good base for C++.
2. C++ is object oriented, allowing for the greatest design flexibility.
3. Microsoft Visual Studio has a framework for direct access to components of the Windows environment.
4. Only a slight trade-off of limiting the end-user base.

5.1.3 Update Notification Design

A significant concern in the implementation of the notification system is the method in which the end-user connects to the information that is stored in the database. There are three possible ways in which a user can connect with the information: directly with a server, directly without a server, or passively.

5.1.3.1 Direct with server

This implementation would involve setting up a client- server architecture whereby a user connects to a server. This server would then notify all active users whenever an update occurred. The major advantage of this implementation is that any notification would immediately be sent to any active users. The disadvantage is that in a potentially large scale design, the server would be connected to many users, required significant hardware. A passive implementation might reduce this need.

5.1.3.2 Direct without server

It is also possible to design a program to connect directly to a database and upon any changes in the database, the application would be notified. The fatal drawback of this system however, is that allowing users to be directly connected to the database could result in security issues, and since the proposed implementation of the notification system is an amber alert system, security is a major factor. Direct access to the database cannot be allowed.

5.1.3.3 Passive

In this implementation, the application itself is responsible for all updates, removing the need for the client server architecture. The application will connect to the web server on regular intervals and check for the latest updates. The only requirement in this case is the web server, which is already functional separately from the application. The major disadvantage of this implementation is that the responsibility of the update is on the end application, and depending on when the application checks, there could be significant delays between an update being ready, and it reaching the end user.

It was decided that the passive implementation would be used in this case due to the indirect connection to the database, the requirement of the web server already having been met, and the fact that the requirements of the system could be met without introducing unneeded complexity.

In summary, it was decided that the alert application would be designed as a visual cue in the form of a pop-up window. It would be written in C++ using the Microsoft Visual Studio design environment. The format of the updates would be passive, requiring a check by the software, and not a signal from the server.

5.2 Design Implementation

The design of the notification is based on the reading of the database table through the website. Figure 5.2 shows the output of one of the display files.

ID	First Name	Middle Initial	Last Name	Message	Date Added	Time Added
152	Taylor	V	Wiebe	Missing	2007-02-28	18:50:33
151	John		Smith		2007-02-27	23:00:23
138	For	Mr.	Shadley		2007-02-20	13:43:43
136	J	C	Lim		2007-02-19	23:09:37
134	Chris	R	Lim		2007-02-19	23:06:09
117	Marty	S	Bucati		2007-02-03	13:42:01
115	Louis	T	Canter	Missing	2007-02-03	13:09:16
74	Kayla	M	Murray	Missing	2007-01-16	18:04:21
73	Jake	M	Pacho	Unknown Location	2007-01-16	18:03:57
72	Ingrid	U	Vladovich	Possible Danger	2007-01-16	18:03:32

Figure 5.2: Display Output

It can be seen that simply reading in the files, separating each entry (parsing), and sorting

based on the unique ID and/or the date would allow a check for new updates. The basic flow could be seen in Figure 5.3.

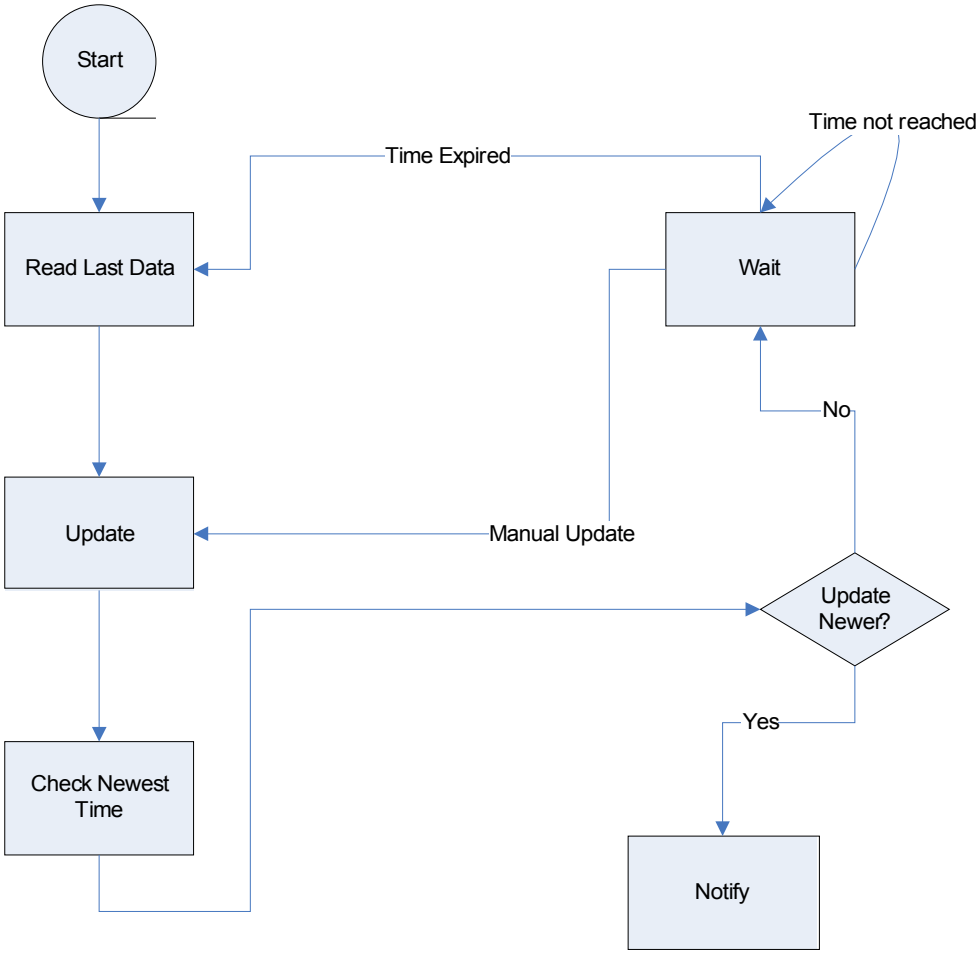


Figure 5.3 Basic Application Flow

When the system starts, it will check for the last entry stored in its local file. If there is no file, it will assume that any entry in the database is new. If there is a file in storage, the last entry is loaded into memory. The system then connects to the website, and downloads the all latest information. This is the content of the full display.php file. It will contain the ten newest entries into the database. If at a later date the display.php file is altered to contain more or less entries, the pop-up application will still function in the same manner. In a simple setup, with only one type of data in the table the program only needs to check the top entry to see if it is different from the last one stored in memory. In a more complex implementation, such as an amber alert table with multiple priority levels and regional information, it would be necessary to check all the entries with the valid tags to distinguish new data. After this data check, a notification can be given in the case of new information, or the system can wait in the case of no new information. The cycle then repeats.

5.2.1 Configuration File/System Initialization

Whenever the program is run, it should check for the user's desired setup options. A flow diagram can be realized as the following:

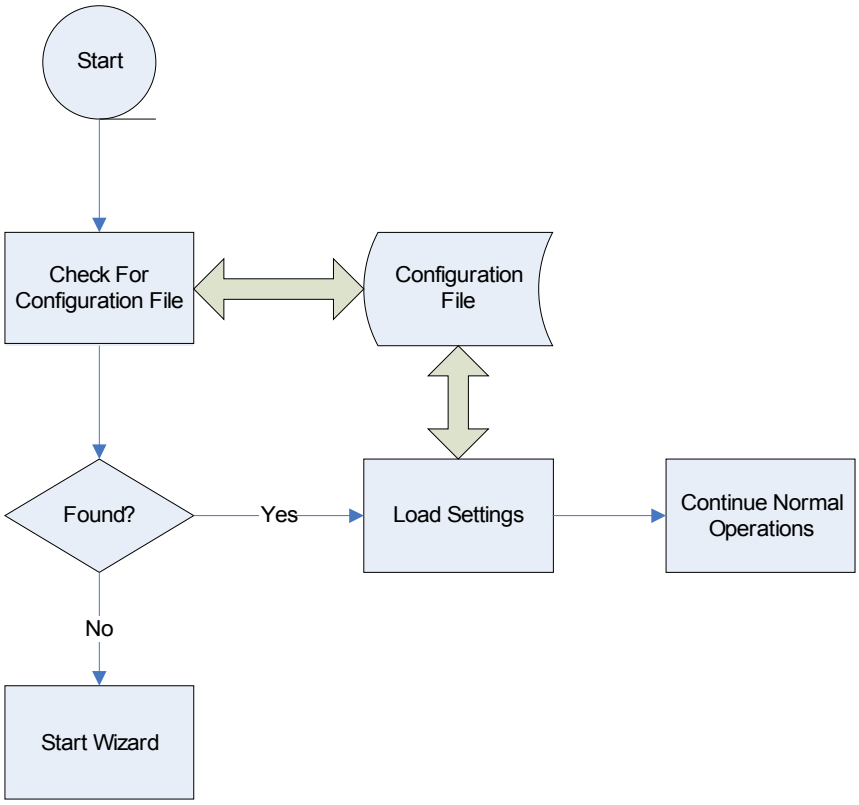


Figure 5.4: Application Initialization

The program should first check for a configuration file. If there is a configuration file stored, it should be loaded into memory and all settings should be active. If there is no file, one should be created, and set up according to the user's preferences. Most computer users, even those with limited experience have previously used a setup "Wizard", so it was decided that this should be the form of the initial user input. A flow chart can be seen in Figure X:

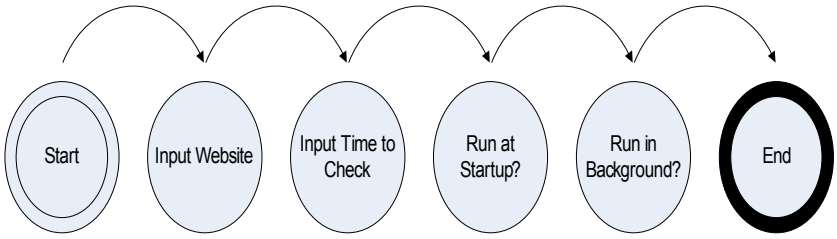


Figure 5.5 Configuration file initialization

This figure shows some of the options that would be important in an amber alert notification program. The “input website” would be the main Internet site of the amber alert organization providing the data. The “data website” would be the data address that is used by the program to check for updates. The “time to check” value could be how often the program is to check for new data. The implications of this “time to check” value will be discussed in a following section. The other two values “run at startup”, and “run in background (minimized)” would be user preferences for the actual program function within their system. It should be noted that for maximum benefit, the program should always be running when the computer is on, hence it should be run on startup. The “run in background” option will be so that the program does not interfere with the users' normal operation.

5.2.2 Connection to Website

The following table shows the methods used in the connection of the program to the website:

InternetOpenUrl	Used to establish a connection to a website URL such as “www.amberthesis.com”
HttpSendRequest	Used to request certain files or directories within a site
InternetReadFile	Used to download a file to a local computer
InternetCloseHandle	Used to close the connection with the web server

Table 5.1: Connection Methods

A flow chart of the connection to the website can be seen in the following figure:

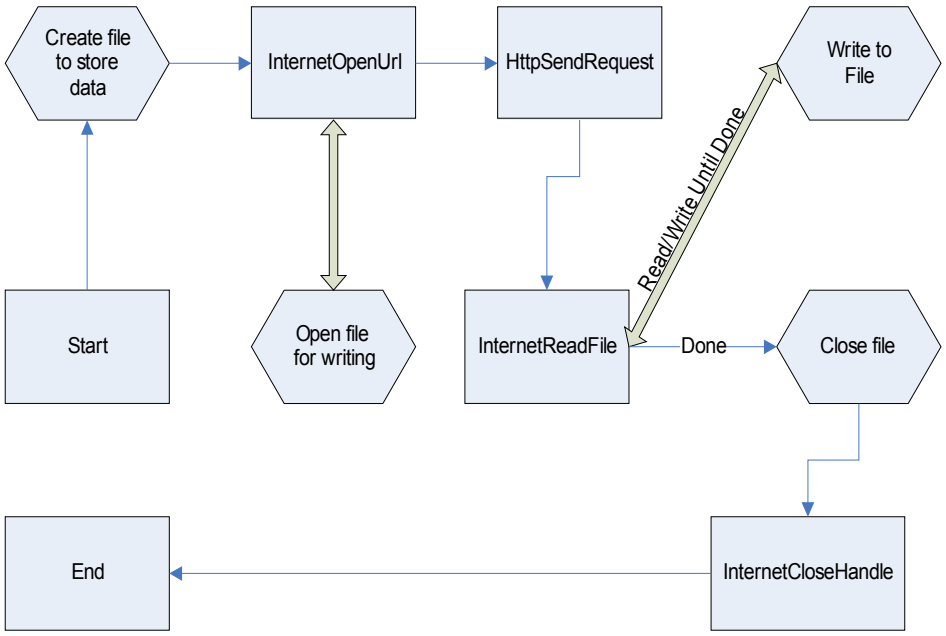


Figure 5.6 Application connection to Website

The program simply goes through each step of the connection process until the desired data has been received. During the "read" phase, the program continues to download data until the entire file has been copied into memory. This is then written into a file. If at any point there is an error, the program will generate an error message, and delay until the next check.

5.2.3 Pop-up generation/handling

There are a few concerns when handling the pop-up generation. The system should check for updates periodically according to the user settings but a user should be able to select to manually check at any time for updates. The pop-up window should also provide the necessary information in the most compact form possible. The following section will discuss these concerns a walk through the basic design of the pop-up generation.

Figure 5.7 shows a flow diagram of the system after initialization.

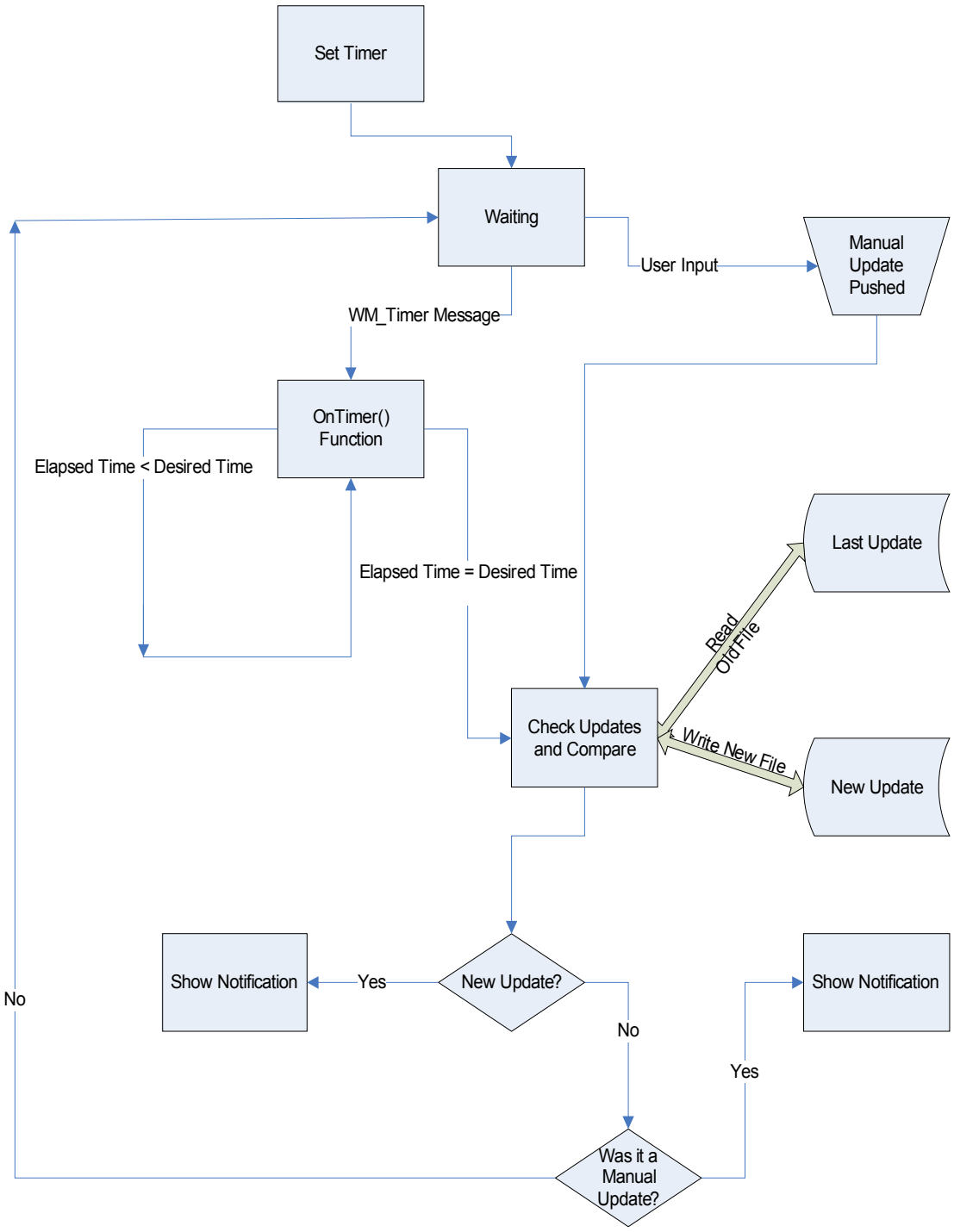


Figure 5.7 Application after initialization

In the initialization of the program, a timer is set. This timer will send a message to the system every time the a default number of seconds has passed. The message generated activates the OnTimer() function. Inside this function, the number of seconds since the last check is tested against the desired time between checks. This value is input by the user in the user interface. If

the two are the same, then the system behaves as though a user has pushed a button for a manual check. Distinguishing between timer checks and a user's manual check will be discussed in the next section. The system then establishes a connection to the web server, and downloads the datafile, writing all data into a local file. The name and location of the file to download can be changed by the user. The data is then compared to the latest stored data. If the downloaded data is newer, the message window class is called with a “New Update” message containing the relevant data. If the data is not newer, then the message window class is called with a “no new updates” message. However, this message should not be sent if this was not a manual check. If the system is working with the timer, there should only be a message if there is an update. If the user manually checks, there should be a message in either case, telling the user of the current findings.

5.2.4 Internal vs External Requests

It was decided that to facilitate possible upgrades to the code, the system should distinguish between user requests and timer conditions without having separate methods with duplicate code. This was accomplished with the token scheme displayed in Figure 5.8.

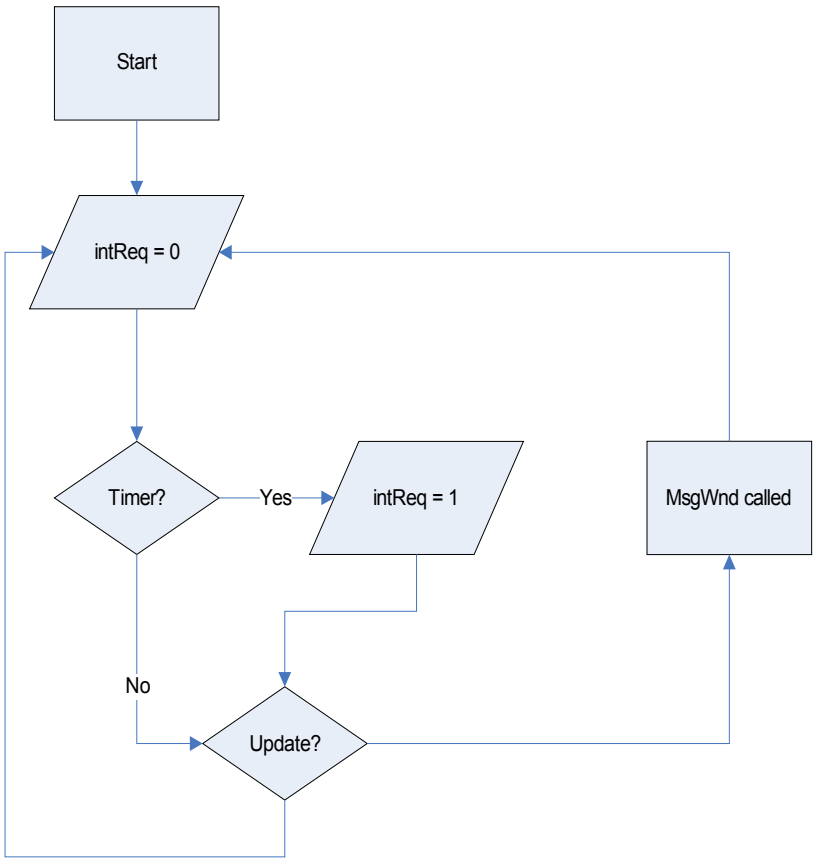


Figure 5.8: Internal vs. External Requests

When the system is initialized, the intReq (internal request) variable is set to "0". This variable is set to "1" if the timer calls the MsgWnd object. At the end of the MsgWnd method, intReq is set to "0" once again. This ensures that a message window will always be shown, unless there are no updates and an internal request has called the method.

5.2.5 User Interface

The graphical user interface (GUI) was implemented as a dialog-based application using Microsoft's Foundation Classes (MFC). The main goal of the user interface was to make the use of the program as simple as possible. It also provides users with the current settings of the application so that they can make any adjustments necessary. The final interface can be seen in Figure 5.9.

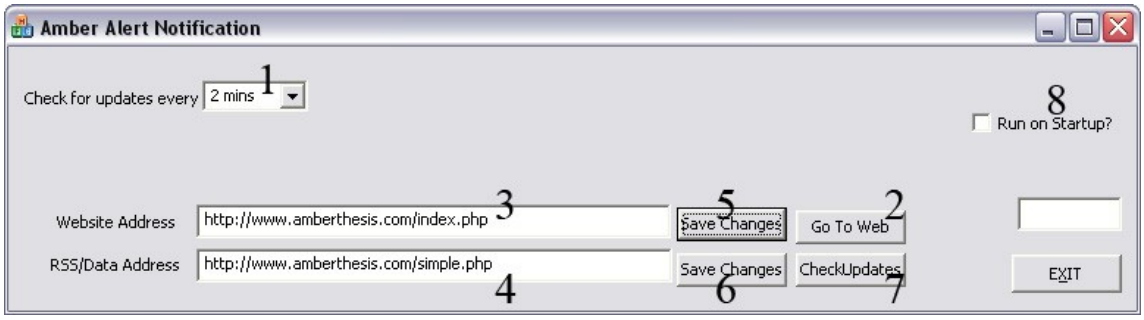


Figure 5.9: The Graphical User Interface

The upper left corner (1) shows where a user can input the amount of time they would like to expire before a check is made. This should be adjustable so that a user doesn't get messages when they do not want them. Also, memory and cpu usage are issues for some users. Ideally though, for a notification program, the program should check as often as possible. Item (2), the "Go to Web" button launches the computer's default Internet browser to the web page located in the website address (3). This value is user configurable and is stored if it is changed, and loaded each time the program starts. Item (4) is the location of the simple data table that is used to check for updates. It functions in the same way as item (3). If either of the "Save Changes" buttons, (5) or (6), are pressed, The user is prompted with a message box such as Figure 5.10.

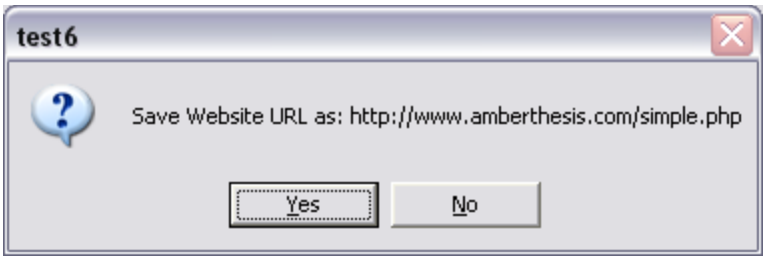


Figure 5.10: A typical message box

If the user selects "Yes" , the changes are written to the configuration file. If the user selects "No", the value is restored to the value that previously occupied the field. Button (7), "CheckUpdates" initializes a manual check for updates. Item (8), the "Run on Startup" was not implemented in this design due to time constraints. The purpose of this check box would be to allow the application to run whenever a user started and logged onto the computer. This would allow a user to be notified without having to remember to start the program each time they logged on. An additional enhancement that was added to the GUI was the option of minimizing to the system tray. When a user clicks on the minimize button (9), the program is minimized as shown:

Figure 5.11: The Notification Application minimized.

This allows a user to enjoy the benefits of the application without seeing it, or having it interfere with other work. The design of the notification window was also changed in implementation from a simple message box, to a message window. When designing the notification, it was found that message boxes could not appear if the application that created the notification was not the topmost window in the system. This would mean that a user would need to click on the application for a notification to appear, nullifying the usefulness of the application. A switch to the message window format meant that the application was functional and also more visually appealing. A comparison of the two styles can be seen in the following figure:

Figure 5.12 Message Box vs Message Window

5.3 Speed/Memory considerations

The application was tested on an AMD Athlon 64 3000+, running at 2.07 Ghz, with 1GB of RAM. Basic testing of the application was divided into two parts, CPU usage, and memory (RAM) usage. The following figure shows the CPU usage before the application is run.

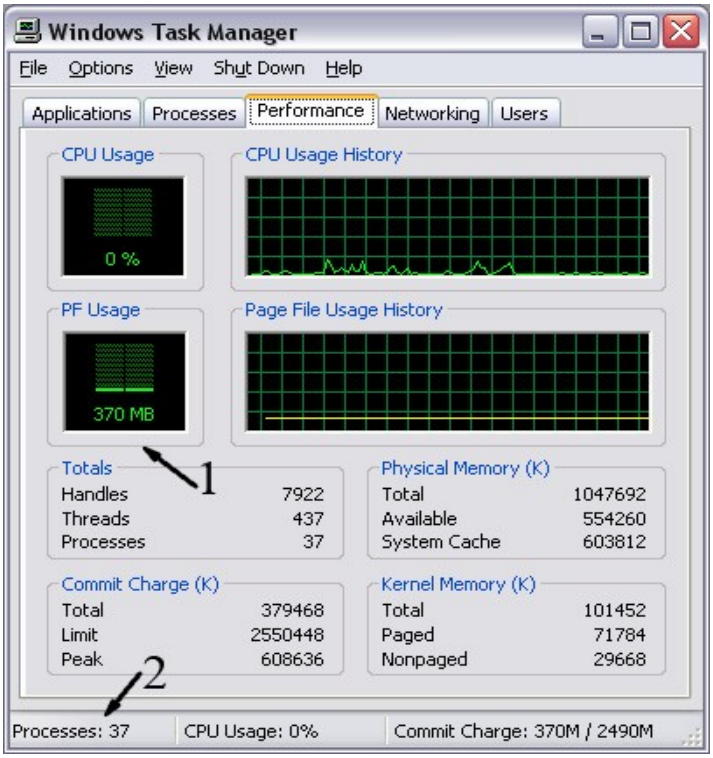


Figure 5.13: CPU usage before application

The system is using approximately 370 MB of RAM (1), and running 37 processes (2). The following screen shot shows the system after the application has been started, including the initial check for updates.

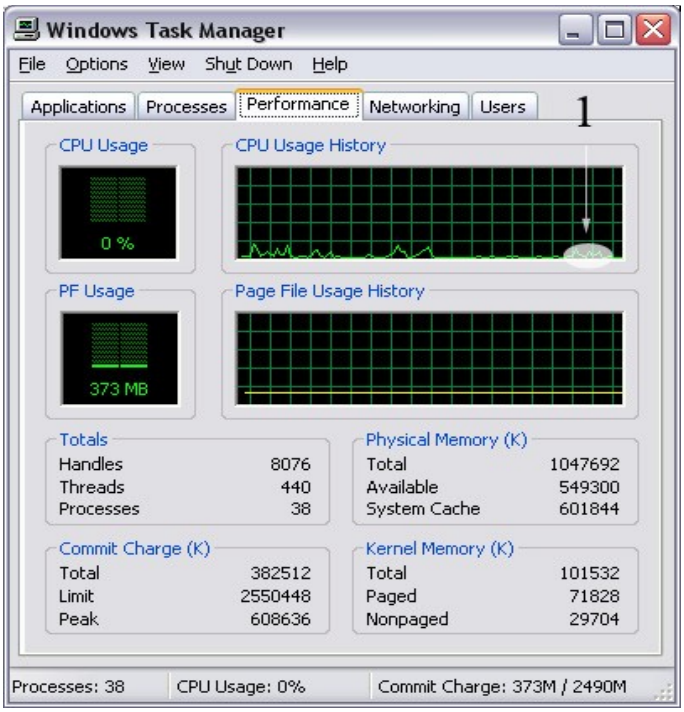


Figure 5.14: CPU usage after application

There is only a slight discernible spike in the CPU usage history after the program is started (1). It was found that due to the small size of the data that was being downloaded (approximately 7kB), the system hardly registers CPU usage during the update phase. Therefore, it was found that a connection and test for updates could be done as often as every 20 seconds. This is almost an instant notification as compared to times on the order of minutes and hours. This means that the functionality of the application can be greatly improved. The second test was done to see memory usage. The memory usage is shown in the Figure X:

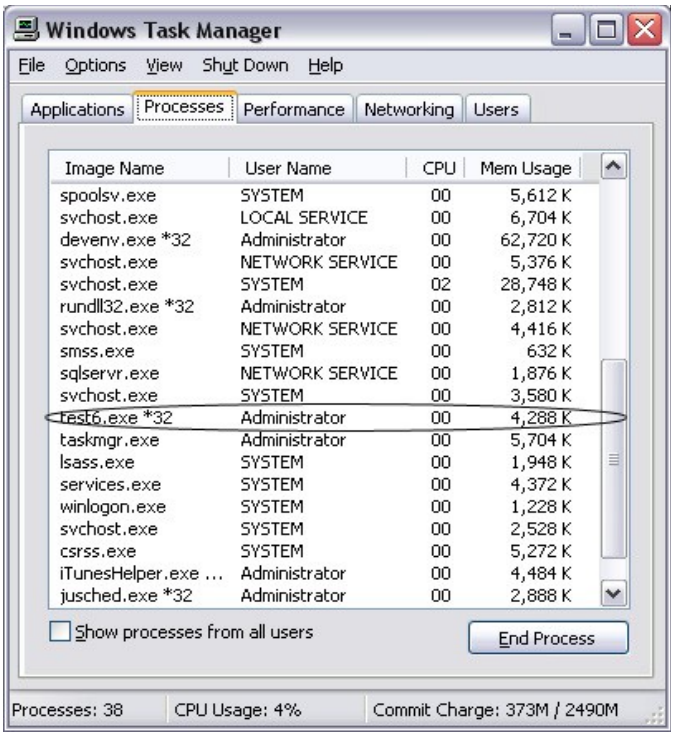


Figure 5.15: Memory Usage

In Computer systems where 512MB of RAM is getting more and more common, 4.2 MB of RAM usage is not a significant amount. Even on a computer with 256MB of RAM, this amounts to less than 2%. This program should be functional on both high speed and economical systems.

Chapter 6 - Conclusion

Many options were considered in the design of the Web Based Notification System. The main goal of this project was to provide a cost efficient and user friendly application for the general public and police officials to be able to interact with in order to spread missing child information. The Linux Apache MySQL PHP bundle proved to be a great choice for such a task. The LAMP platform reduced cost of implementation significantly and still provided a robust environment to develop a user friendly web application. The system showed great support for portability to implement itself with other existing databases by using the ODBC standard.

The notification application provided a practical form of alerting a user working on a desktop computer. The programs ability to hide within the task bar and generate an alert window made it feasible to have as a constantly running application hidden from the users view. The embedding of PHP scripting into HTML pages created a user friendly environment and easy to navigate web site that serves its functionality by giving the public the latest update on Amber Alerts. Using a passive, Internet connection based form of checking allowed the focus to be put on the application functionality and ease of use, rather than the design of a complex client/server architecture. System testing proved that CPU and memory usage were not an issue, and that the application could check very often without slowing down the computer.

Overall, the design implementations proved to generate a feasible system that can be used in emergency situations and would help in the communication of important information.With the expanding use of the Internet, a web based application will be a great communication tool in transmitting life-threatening emergencies such as Amber Alerts. This system hopes to contribute to the communication of important data and hopes to eventually aid in the saving of lives.

6.1 Future Work and Considerations

Many features were considered when designing the Web Based Notification system. However, due to time constraints, some of these features could not be fully realized. This chapter will discuss future work that can be done to further enhance the Web Based Notification system. Such features include dynamic page generation, an E-mail service from the web server, uploading files to the database and additional functions to the application.

6.1.1. Dynamic Page Generation

Currently, missing child information is displayed on the main page in the form of a table. The main goal was to have each entry within that table to be a link to a full dedicated web page to

that missing child. This would be achieved through a template web page that would take in specific variables in order to dynamically render a web page to display more information about a specific child. Upon selecting a child entry in the displayed table, each entry would link to the same template web page. The key difference is that each child within the table would pass on its own key variable which would be used to query the database and retrieve information specific only to that child and display that information in anyway desired. The template page would display all information on the child and display media as well including images, video, and maps. A link to contact officials regarding that specific child would also be provided.

6.1.2 Email Service

An Email server would be integrated and built into the main web server in order to allow an Email service from the web site. Users would receive an email confirming their registration and displaying their information.

The email service can be used to send out forgotten passwords when logging in and provide updates to users about new versions of the notification application. If desired, an email can be sent out to all users upon an Amber Alert to provide users with a record of the event in the case that a user's workstation is not online at the moment. A function has been written into register.php called sendMail. This function takes in the user's id and password as parameters. What sendMail does is simply concatenate a series of text strings that include a user's specific information retrieved upon registration and generates a proper greeting and confirmation letter. The message is then emailed to the user using the PHP command mail. Mail takes in the users email as one parameter, and an arbitrary text string to serve as a topic, as another parameter.

Although this function was written and integrated into the registration procedure, the function was not called. This was due to the web server not having an E-mail server integrated. Because of time constraints, an E-mail server was not installed and development within this area was at a halt. Future work would include additional written email functions which embed other scripts and the rebuilding of the web server with a built in email server.

6.1.3 Uploading media files to database

An option to upload media files such as images and video files to the database was put into the admin.php script. Unfortunately the necessary work to transform the uploaded file to BLOB format in SQL was not completed due to time restraints.

6.1.4 Notification Application Enhancements

For future work, the application could be set up to run every time the computer is turned on, allowing a user to use the system without having to interact constantly. Also, the application

could be rewritten using another programming language to provide multi-platform support, using Java, or a managed application, using C# or J#.

References

[CFM07] Child Find Manitoba Web Site, “Manitoba Amber Alert Plan”, [March 1st, 2007], Available at HTTP: http://childfind.mb.ca/en/missing_children/amber_alert/

[D307] Data 360, “Global Internet Usage”, [March 1st, 2007], Available at HTTP: http://www.data360.org/graph_group.aspx?Graph_Group_Id=315

[DB2EC07] DB2 Express-C Web Site, “What is DB2 Express-C?”, [March 1st, 2007], Available at HTTP: http://www.306.ibm.com/software/data/db2/express/getstarted.html

[Dirk04] LINC Project, “Microsoft Windows”, [March 1st, 2007], Available at HTTP: http://www.lincproject.org/toolkit/cos_guide/operating_system/windows

[IBMDB207] IBM Software Web Site, “DB2 Product Family”, [March 1st, 2007], Available at HTTP: http://www-306.ibm.com/software/data/db2/

[IWS07] Internet World Stats, “Internet Usage Statistics for the Americas”,[March 1st, 2007], Available at HTTP: http://www.internetworldstats.com/stats2.htm

[Mad06] Madden, Mary “Internet Penetration and Impact April 2006”, Data Memo for PEW/Internet, Pew Internet & American Life Project, 2006. [March 1st, 2007], Available at HTTP: http://www.pewinternet.org/pdfs/PIP_Internet_Impact.pdf

[MySQL07] MySQL Web Site, “Why MySQL?”, [March 1st, 2007], Available at HTTP: http://www.mysql.com/why-mysql/

[ODBC07] MySQL5.1 Reference Manual, “25.1.1.2.2. ODBC Driver Managers”, [Online document], [March 1st, 2007], Available at HTTP: http://dev.mysql.com/doc/refman/5.1/en/myodbc-general-information.html

[Ram05] Ramana, U.V. and Prabhakar, T.V., “Some experiments with the performance of LAMP architecture”, Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on, pp. 916 – 920, 21-23 Sept. 2005

[Sun07] Sun Microsystems Commercial Website: About Java Technology, [March 1st, 2007], Available at HTTP: http://www.sun.com/java/about/

[Wik07] Wikipedia: C++, [March 1st, 2007], Available at HTTP: http://en.wikipedia.org/wiki/C++

Appendix A – AlertDlg.cpp

```
// AlertDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Alert.h"
#include "AlertDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();


// Dialog Data
	enum { IDD = IDD_ABOUTBOX };


	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()


// CAlertDlg dialog




CAlertDlg::CAlertDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CAlertDlg::IDD, pParent)
	, m_inurl(_T(""))

	, m_timer(_T(""))
	, m_time(_T(""))
	, mWebAddr(_T(""))
	, internReq(0)
	, m_bMinimized(false)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CAlertDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_EDIT3, m_inurl);
```

```
	DDX_Text(pDX, IDC_EDIT5, m_timer);
	DDX_CBString(pDX, IDC_COMBO1, m_time);
	DDX_Text(pDX, IDC_EDIT1, mWebAddr);
}

BEGIN_MESSAGE_MAP(CAlertDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_WEB, &CAlertDlg::OnBnClickedWeb)
	ON_BN_CLICKED(IDC_BUTTON1, &CAlertDlg::OnBnClickedButton1)
	ON_BN_CLICKED(IDC_UPDATE, &CAlertDlg::OnBnClickedUPDATE)
	ON_BN_CLICKED(IDOK, &CAlertDlg::OnBnClickedOk)
	ON_WM_TIMER()
	ON_CBN_SELCHANGE(IDC_COMBO1, &CAlertDlg::OnCbnSelchangeCombo1)
	ON_BN_CLICKED(IDC_SaveWeb, &CAlertDlg::OnBnClickedSaveweb)
	ON_MESSAGE(WM_TRAY_MESSAGE,OnTrayNotify)
	ON_BN_CLICKED(IDC_Minimize, &CAlertDlg::OnBnClickedMinimize)
END_MESSAGE_MAP()


// CAlertDlg message handlers

BOOL CAlertDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	SetupMinimizeToTray();

	//input default addresses
	m_inurl = "http://www.amberthesis.com/~limmer21/simple.php";
	mWebAddr= "http://www.amberthesis.com/~riceqube/index.php";

	//set internal request token
	internReq=0;

	//m_timer = "timer for 100";
	m_time = "2 mins";

	//used for edit boxes
	UpdateData(FALSE);

	//create a new timer
	CompTime =CTime::GetCurrentTime();
	utimer = SetTimer(IDT_TIMER, 1000, NULL);
	if (utimer==0){
		AfxMessageBox("Error, timer not created");
```

```
	}

	mTimerInterval=20;



	//start minimized>
	int nResult=AfxMessageBox("Minimize Program Now? " , MB_YESNO|MB_ICONQUESTION);
		 if(nResult == IDYES)
		{
			CAlertDlg::ShowWindow(SW_SHOWMINIMIZED);
		}

	//do initial check
		 CAlertDlg::OnBnClickedButton1();







	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CAlertDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CAlertDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()),
0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CAlertDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
```

```
}


void CAlertDlg::OnBnClickedWeb()
{
	ShellExecute(NULL,NULL,(LPCSTR)mWebAddr,NULL,NULL,SW_SHOWNORMAL);
	CAlertDlg::ShowWindow(SW_SHOWMINIMIZED);
	// TODO: Add your control notification handler code here
}



void CAlertDlg::OnBnClickedButton1()
{
	HINTERNET Initialize,File;
	DWORD dwBytes;
	char ch;
	FILE *fp;
	FILE *fp2;
	char filename[255];
	char filename2[255];
	char * cptr;
	char * mPCase;
	char buffer[256];
	char buffer2[256];
	int popHeight=100;
	int popWidth=100;
	char *mMsg="No New Updates";



	SetCursor(LoadCursor(NULL,IDC_WAIT));
	Initialize = InternetOpen("HTTPGET",INTERNET_OPEN_TYPE_DIRECT,NULL,NULL,0);
	if ( !Initialize )
	{
	AfxMessageBox("InternetOpen Failed");
	return;
	}


	//open up an HTTP request
	File = InternetOpenUrl(Initialize,(LPCSTR)(CAlertDlg::m_inurl), NULL, 0, 0, 0);

	//open file to write to
	sprintf_s(filename,"data/datafile.txt");

		if ((fp = fopen(filename, "w")) != NULL)
		{

			if(HttpSendRequest(File,NULL,0,NULL,0))
			{
				while(InternetReadFile(File,&ch,1,&dwBytes))
				{
				if(dwBytes != 1)break;

				putc(ch,fp);

				}
			}

		}
		fclose(fp);

		if ((fp = fopen(filename, "r")) != NULL)
		{
			 cptr = fgets(buffer, 256, fp);
			 sprintf_s(filename2,"data/latestCase.txt");
			 if((fp2 =fopen(filename2,"w"))!=NULL)
			 {
				 fprintf(fp2,"%s",cptr);
			 }
			 fclose(fp2);
```

```
                        //AfxMessageBox(cptr);
                }
                fclose(fp);

                sprintf_s(filename,"data/previousCase.txt");
                if((fp=fopen(filename,"r+"))!=NULL)
                {
                        mPCase= fgets(buffer2,256, fp);

                        if(mPCase!=NULL){

                                if(strcmp(mPCase,cptr)!=0){
                                        //AfxMessageBox("New Child Missing: " +
(CString)cptr);
                                        //AfxMessageBox("still working");

                                        /*if(t_MsgWnd->IsOpen()==true){
                                                AfxMessageBox("test message");
                                                mMsg="Multiple Updates";
                                                t_MsgWnd->ChangeText(mMsg);

                                        }
                                        else{*/
                                                //mMsg="New Child Missing: " , (CString)cptr;


                                        //AfxMessageBox("New Child Missing: " +
(CString)cptr,MB_ICONINFORMATION|MB_OK);
                                        t_MsgWnd = CMsgWnd::CreateObject(
                                        _T("New Child Missing: " + (CString)cptr), // the
message to be displayed
                                        180, // width of the window
                                        150, // height of window
                                        4000, // time for the message to be displayed
                                        5, // delay for animation, how fast the window opens
and closes
                                        CRect(30, 30, 130, 110), // rectangle in the window
where the

//message will be displayed
                                        this // parent of the message window
                                        );
                                        //if(t_MsgWnd!=NULL){
                                                t_MsgWnd->PopMsg();
                                        //}




                                //(MessageBox("New Child Missing: " +
(CString)cptr))ShowWindow(SW_RESTORE);
                                        fclose(fp);
                                        if((fp=fopen(filename,"w"))!=NULL)
                                        {
                                                fprintf(fp, "%s", cptr);
                                        }
                                        fclose(fp);
                                //}
                                }
                                //else if(internReq==0)
                                //{

                                        t_MsgWnd = CMsgWnd::CreateObject(
                                        _T("No New Updates"), // the message to be displayed
                                        180, // width of the window
                                        150, // height of window
                                        4000, // time for the message to be displayed
                                        5, // delay for animation, how fast the window opens
and closes
                                        CRect(30, 30, 130, 110), // rectangle in the window
where the
```

```
//message will be displayed
                                        this // parent of the message window
                                        );
                                        t_MsgWnd->PopMsg();

                                //}
                        }
                        else
                        {
                                fprintf(fp, "%s", cptr);
                                fclose(fp);
                        }
                }
                //fclose(fp);




        /*close file , terminate server connection and
        deinitialize the wininet library*/
        InternetCloseHandle(File);
        //InternetCloseHandle(Connection);
        InternetCloseHandle(Initialize);
        SetCursor(LoadCursor(NULL,IDC_ARROW));

}




void CAlertDlg::OnBnClickedUPDATE()
{
        CString m_oldURL=m_inurl;
        FILE *fp;
        char filename[255];

        sprintf_s(filename,"data/url.cfg");

        UpdateData(TRUE);
        int nResult=AfxMessageBox("Save Website URL as: " + m_inurl, MB_YESNO|
MB_ICONQUESTION);
                 if(nResult == IDYES)
                {
                        if ((fp = fopen(filename, "w")) != NULL)
                        {
                                fprintf(fp,"%s",m_inurl);
                        }
                        fclose(fp);
                   //save to file
                }
                else
                {
                        m_inurl=m_oldURL;
                        UpdateData(FALSE);
                  //reset to original
                }
}
void CAlertDlg::OnBnClickedOk()
{
        KillTimer(IDT_TIMER);
        //AfxMessageBox("Timer Killed");
        OnOK();
}

void CAlertDlg::OnTimer(UINT_PTR nIDEvent)
{
        //AfxMessageBox("timer");
        CTime CurTickValue = CTime::GetCurrentTime();
```

```
	CTimeSpan Difference = CurTickValue - CompTime;

	//m_timer.Format("%d",(Difference.GetTimeSpan())%mTimerInterval);
	//UpdateData(FALSE);

	if(((Difference.GetTimeSpan())%mTimerInterval)==0){
		//AfxMessageBox("time");
		internReq=1;
		CAlertDlg::OnBnClickedButton1();
	}

	CDialog::OnTimer(nIDEvent);
}

void CAlertDlg::OnCbnSelchangeCombo1()
{
	//UpdateData(TRUE);
	if(m_time.Compare("2 mins")==0){
		mTimerInterval = 120;
	}
	else if(m_time.Compare("5 mins")==0){
		mTimerInterval = 300;
	}

	else if(m_time.Compare("30 mins")==0){
		mTimerInterval = 1800;
	}
	else if(m_time.Compare("1 hour")==0){
		mTimerInterval = (60*60);
	}


}



void CAlertDlg::OnBnClickedSaveweb()
{
	CString mOldAddr=mWebAddr;
	FILE *fp;
	char filename[255];

	sprintf_s(filename,"data/webAddr.cfg");

	UpdateData(TRUE);
	int nResult=AfxMessageBox("Save Website URL as: " + mWebAddr, MB_YESNO|
MB_ICONQUESTION);
		 if(nResult == IDYES)
		{
			if ((fp = fopen(filename, "w")) != NULL)
			{
				fprintf(fp,"%s",mWebAddr);
			}
			fclose(fp);
		   //save to file
		}
		else
		{
			mWebAddr=mOldAddr;
			UpdateData(FALSE);
		  //reset to original
		}
}

afx_msg LRESULT CAlertDlg::OnTrayNotify(WPARAM wParam, LPARAM lParam)
  {
    UINT uID;
    UINT uMsg;

    uID = (UINT) wParam;
    uMsg = (UINT) lParam;
```

```
    if (uID != 1)
      return true;

    CPoint pt;


    switch (uMsg )
    {

    case WM_LBUTTONUP:
      GetCursorPos(&pt);
      ClientToScreen(&pt);
      OnTrayLButtonDown(pt);
      break;

    case WM_RBUTTONDOWN:
    case WM_CONTEXTMENU:
      GetCursorPos(&pt);
      OnTrayRButtonDown(pt);
      break;

    }
    return true;
  }

void CAlertDlg::OnTrayLButtonDown(CPoint pt)
    {
          pt = pt; // Prevents compiler warnings
    MaximizeFromTray();
    }

void CAlertDlg::OnTrayRButtonDown(CPoint pt)
    {
      //m_menu is the member of CTrayMinDlg as CMenu m_menu;
      m_menu.GetSubMenu(0)->TrackPopupMenu(TPM_BOTTOMALIGN|
       TPM_LEFTBUTTON|TPM_RIGHTBUTTON,pt.x,pt.y,this);
    }

void CAlertDlg::SetupMinimizeToTray()
{
       //AfxMessageBox("setup minimize");
    m_TrayData.cbSize = sizeof(NOTIFYICONDATA);
    m_TrayData.hWnd  = this->m_hWnd;
    m_TrayData.uID = 1;
    m_TrayData.uCallbackMessage  = WM_TRAY_MESSAGE;
    m_TrayData.hIcon = this->m_hIcon;

    strcpy(m_TrayData.szTip, "test6");
    m_TrayData.uFlags = NIF_ICON|NIF_MESSAGE|NIF_TIP;

    BOOL bSuccess = m_menu.LoadMenu(IDR_MENU1);
    if (!(bSuccess))
    {
        MessageBox("Unabled to load menu", "Error");
    }

    m_bMinimized = false;
}

void CAlertDlg::MinimizeToTray()
{
       //AfxMessageBox("minimizeToTray");
    BOOL bSuccess = Shell_NotifyIcon(NIM_ADD, &m_TrayData);
    if (!(bSuccess))
    {
        MessageBox("Unable to set tray icon", "Error");
    }

    this->ShowWindow(SW_MINIMIZE);
    this->ShowWindow(SW_HIDE);
```

```
    m_bMinimized = true;
}

void CAlertDlg::MaximizeFromTray()
{
    this->ShowWindow(SW_SHOW);
    this->ShowWindow(SW_RESTORE);

    Shell_NotifyIcon(NIM_DELETE, &m_TrayData);

    m_bMinimized = false;
}

void CAlertDlg::OnBnClickedMinimize()
  {
       this->ShowWindow(SW_MINIMIZE);
       this->MinimizeToTray();
}

void CAlertDlg::OnSize(UINT nType, int cx, int cy)
{
    if (nType == SIZE_MINIMIZED)
    {
        MinimizeToTray();
    }
    else
    {
        CDialog::OnSize(nType, cx, cy);
    }
}
void CAlertDlg::OnDestroy()
{
    CDialog::OnDestroy();

    if (m_bMinimized == true)
    {
        Shell_NotifyIcon(NIM_DELETE, &m_TrayData);
    }

    // Unload the menu resource
    m_menu.DestroyMenu();
}
```

Appendix B – loggedin.php

```
<?php
        header("Expires: Thu, 17 May 2001 10:17:17 GMT");    // Date in the past
        header ("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT"); // always modified
        header ("Cache-Control: no-cache, must-revalidate");  // HTTP/1.1
        header ("Pragma: no-cache");                          // HTTP/1.0
        session_start();

        if (!isset($_SESSION['SESSION'])) require ( "../../include/session_init.php");

        // reset session variables...
        $_SESSION['USERID'] = "";
        $_SESSION['LOGGEDIN'] = false;
        $_SESSION['EMAIL'] = "";
        $_SESSION['FNAME'] = "";
        $_SESSION['LNAME'] = "";

        // initialize variables...
        $userid = "";
        $passwd = "";
        $email = "";

        // make sure post parameters were sent...
        if (isset($HTTP_POST_VARS["userid"])) $userid =
addslashes($HTTP_POST_VARS["userid"]);
        if (isset($HTTP_POST_VARS["passwd"])) $passwd =
addslashes($HTTP_POST_VARS["passwd"]);

        $_SESSION['USERID'] = $userid;

        // form variables must have something in them...
        if ($userid == "" || $passwd == "" || !is_numeric ($userid)) { header("Location:
../login_system.php?flg=red&userid=".$userid); exit; }

        // check in database...
         $query = "SELECT * FROM tbl_users WHERE iUserID = ".$userid;

         //echo $query;
          $conn=odbc_pconnect($_SESSION['ODBC_DSN1'],$_SESSION['ODBC_LOGIN1'],$_SESSION['
ODBC_PASS1'])
                  or die("Unable to connect to SQL server");
          //mysql_select_db($_SESSION['MYSQL_DB1']) or die("Unable to select database");
               $result = odbc_exec($conn,$query) or die("Invalid query: " .
odbc_error());

               // if userid is not present in DB go back to login page...
               if (odbc_fetch_row($result) != 1) { header("Location:
../login_system.php?flg=red&userid=".$userid);; exit; }

               // check for password, active state, user type, and then send to
appropriate section...
               if ($row['sPassword'] = odbc_result($result, "sPassword")) {
                       // echo $row['sPassword'] . "<br>" . md5($passwd);
                       if (strcmp($row['sPassword'], md5($passwd)) != 0)
{ header("Location: ../login_system.php?flg=red&userid=".$userid); exit; }

                       // set standard session variables...
                       $_SESSION['LOGIN_TYPE'] = $user_type;
                       $_SESSION['USERID'] = $userid;
                       $_SESSION['EMAIL'] = $email;
                       $_SESSION['LOGGEDIN'] = true;
                       $_SESSION['FNAME'] = $row['sFName'];
                       $_SESSION['LNAME'] = $row['sLName'];

                       header("Location: ../account.php");
                       exit;
```

```
                } else {
                        header("Location: ../login_system.php?flg=red&userid=".$userid);
exit;
                }

?>
```

Vita

Name:	Shadley Davidson
Place of Birth:	Cape Town, South Africa
Year of Birth:	1984
Secondary Education:	John Taylor Collegiate (1998-2002)
Awards:	Deans Honour List (2003)
	Hogg Centennial Entrance Scholarship

Name:	Christopher Lim
Place of Birth:	Winnipeg, Manitoba
Year of Birth:	1983
Secondary Education:	Glenlawn Collegiate (1998-2001)
Awards:	Guertin Centennial Entrance Scholarship

Name:	Oliver Tran
Place of Birth:	Winnipeg, Manitoba
Year of Birth:	1982
Secondary Education:	St. James Collegiate (1996-2000)
Awards:	Red River Writing Competition Award
	St. James Student Alumni Award