

EMS Portable Workflow

A Major Qualifying Project
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

Brian R. L'Heureux

Michael J. McHugh

Benjamin D. Privett

Date: March 4, 2003

Approved:

Professor Robert E. Kinicki, Major Advisor

Professor Emmanuel O. Agu Co-Advisor

Abstract

This Major Qualifying Project automates the process by which Worcester Polytechnic Institute's Emergency Medical Services department handles their incident reporting. The system consists of a PDA-based component that sends information over WPI's wireless network to a server-side component for storage in a database. The client-side component is written in C for the Palm OS and uses TCP/IP and 802.11b to communicate with the server. The server, written in Java, uses JDBC to insert the received data into a MySQL database.

Table of Contents

Abstract	2
Table of Contents	3
Table of Figures	5
1 Introduction	6
2 Literature Review	8
2.1 Introduction	8
2.2 Emergency Medical Services	8
2.2.1 Existing Portable EMS Systems	9
2.3 Medical Information Regulations	10
2.4 Palm OS History and Overview	12
2.5 Sony Clié PEG-NX60	14
2.5.1 PEGA – WL100	15
2.6 Client-Side Data Entry	15
2.6.1 User Interface	15
2.6.2 Operating System	16
2.6.3 Palm Basics	17
2.6.4 Palm Applications	18
2.6.5 Formatting Data for Transmission	19
2.7 Data Transmission	20
2.7.1 Direct Serial Access	20
2.7.2 Wireless Transmission	20
2.7.3 Security	27
2.8 Server-side Processing	30
2.8.1 Data Processing Server	30
2.8.2 Database Implementation	31
3 Methodology	35
3.1 Introduction	35
3.2 Client-Side Application	36
3.2.1 User-Friendliness	36
3.2.2 Emergency Incident Reporting Design	38
3.3 Data Transmission	46
3.3.1 Wireless Transmission	47
3.4 Server-Side Processing	53
3.4.1 Data Processing Server	53
3.4.2 Database Implementation	55
3.5 Design Conclusion	57
4 Implementation	57
4.1 Client Application	57
4.1.1 Application Organization	57
4.1.2 Application Operation Overview	57
4.1.3 Event Driven Application Execution	58
4.1.4 Writing to the Palm Databases	63
4.2 Data Transmission	68
4.2.1 Wireless Transmission (NetLib)	68
4.3 Server Application	75
4.3.1 Managing settings	75
4.3.2 Handling connections	77
4.3.3 Managing Responders	77
4.3.4 Parsing XML Data	78

4.3.5	Database Insertion.....	80
4.3.6	Emailing Incident Reports	81
4.3.7	Providing Client Status	81
4.3.8	Reporting	82
5	Testing.....	84
5.1	Testing Methodology	84
5.2	Results/Concerns.....	84
6	Future Research.....	85
6.1	Conduit Development	85
6.2	Adding Signature Functionality.....	86
6.3	Storing/Sending Multimedia	86
7	Conclusion	87
8	References.....	89
9	Appendix.....	92
9.1	Incident Reports	92
9.1.1	Norwood Call Sheet	92
9.1.2	WPI's Existing Call Sheet	93
9.2	Sony Clié NX60.....	94
9.2.1	NX60.....	94
9.2.2	Sony Clié Launcher.....	95
9.3	Server Design Information	96
9.4	EMS Acceptance Letter	98
9.5	Users' Manual.....	99
9.5.1	Installing Application Software	99
9.5.2	Using the EMS Application on the Handheld.....	104
10	Server Installation Guide.....	117
10.1	Server System Requirements	117
10.2	Server Application.....	117
10.3	Database Management System.....	117
10.4	Web Interface	118
10.5	Access Interface.....	119
11	Server Functions.....	119
11.1	Managing Responders.....	119
11.2	Viewing Reports	120
11.3	Editing Data	120

Table of Figures

Figure 1: The Sony Clié PEG-NX60 (SONY03)	14
Figure 2: The Ethernet Cradle Connection Process	14
Figure 3: The PEGA - WL100	15
Figure 4: The Graffiti Area	17
Figure 5: Palm access frequency.....	17
Figure 6: Direct Serial Access Connectivity.....	20
Figure 7: Wireless Connection Process of a PDA (WINT01).....	21
Figure 8 WPI's Wireless Network	22
Figure 9: The Network Protocol Stack (WINT01)	23
Figure 10: High Level System Flow	35
Figure 11: High Level Client Side Diagram	39
Figure 12: Main Menu Form.....	42
Figure 13: An Input Form.....	43
Figure 14: A Text Field	44
Figure 15: A Selector	45
Figure 16: Push Buttons	45
Figure 17: Checkboxes.....	45
Figure 18: Database Control Form.....	46
Figure 19: Database Entry Form.....	46
Figure 20: Connection Process.....	47
Figure 21: Sending an Incident Report	49
Figure 22: Receiving the Responder List	50
Figure 23: Server Overview	53
Figure 24: Files loaded by the configuration manager.....	76
Figure 25 Web Interface Main Menu	82

1 Introduction

Personal Digital Assistant (PDA) technology has grown considerably since it was first created in 1996, allowing users to organize their personal and professional lives in a simplistic and portable fashion. Many of these users rely on their handheld devices to perform tasks such as keeping track of schedules and appointments, managing data such as address books and memo notes, and recently, communicating via peer-to-peer networks¹. These powerful devices provide an ease of use and convenience desired by users. With their increase in popularity and demand, PDA technologies have needed to keep up with the ever changing technologies around them.

The rapid growth of both wireless networking and handheld systems are allowing people to communicate with greater convenience. Evolution in peer-to-peer networks has given rise to more possibilities for handheld computer communication. A virtually untapped technology, wireless connectivity by way of a PDA gives users the ultimate advantage for communication; that is, they can communicate over the Internet without being confined to a single point of access.

Personal Digital Assistants already play a role in the day to day operations of the Emergency Medical Services (EMS). Work schedules can be managed, medication information can be accessed, and medical texts can be stored and accessed through the use of handheld devices (STRE02). As current technologies stand, no reasonably affordable system exists that allows EMS technicians to transfer data from a handheld device by way of a wireless medium.

Prior to the implementation of the EMS Portable Workflow System, the Worcester Polytechnic Institute EMS group, like most EMS groups, still did its most common task of incident reporting on paper. In automating this task, the job of the Emergency Medical Technician (EMT) is simplified. Rather than carrying around a clipboard and a stack of forms, they can now carry a small PDA. The

¹ As opposed to the client-server model, there are no fixed clients and servers in a peer-to-peer network (TANE03).

goal of this MQP is to provide a service for the EMS group that would make their extremely important jobs easier.

Using the new system, EMTs enter incident information on a PDA. This information is stored until a wireless network connection is established, at which point it is sent to a server. The server is responsible for storing the incident data in a database that is accessible for EMS reporting needs and sending the incident data via email to specified users.

This system has unique capabilities in the current state of EMS portable technology that provide the EMS technician with the ability to submit an incident report over a wireless network. Utilizing the most recent advances in handheld technology, the system reduces the amount of administrative overhead required to perform the daily tasks of the EMS department.

2 Literature Review

2.1 Introduction

For every incident an EMS technician responds to, they are required to fill out an incident report. Prior to implementation of the EMS Portable Workflow System, the technicians carried paper reports from incident to incident. At the end of his or her shift, each technician was required to submit one report per incident for later manual entry into a database. The EMS Portable Workflow System simplifies this process by allowing the reports be filed and submitted remotely with a PDA to a centralized database, thus eliminating the arduous task of manual data entry.

The EMS Portable Workflow System utilizes the standard client-server model. The use of a variety of technologies was necessary for successful completion of the project. Client-side development centered on development of the handheld application, which included creation of the application itself, user interface issues, and the storage of data in a handheld database. To transmit data to a database over a wireless network, current techniques were researched to determine the most appropriate means of transmission of data in the system. Additionally, to develop the server-side component, research was conducted on different server and database technologies. Each technology utilized in this project is discussed in the sections that follow.

2.2 Emergency Medical Services

Currently, WPI's EMS group is one of the approximately two hundred campus-based EMS groups (SMIT01). The WPI EMS group is responsible for first-response medical service on the WPI campus. EMS is managed by the WPI Campus Police Department and employs students as technicians. Each technician is certified at the Massachusetts First Responder level, and is trained to handle most medical emergencies. Each incident report contains information about the patient that can be used later by EMS or any other group that may need to give aid to that patient. The report contains such information as the patient's status upon the technician's arrival, a brief medical history, present

medications, allergies, measurements taken at the site of the incident, and the responder(s) to the incident. Two incident reports were referenced in this project. The first is the Norwood Call Sheet which is the incident report form that the Emergency Incident Reporting application is modeled after. The second is the existing WPI EMS incident report form. See section 9.1 in the appendix for the incident reports.

2.2.1 Existing Portable EMS Systems

The EMS Portable Workflow System is unique. It includes wireless synchronization capability, and integration with a backend database. Most of the existing systems are either cost-prohibitive for a smaller EMS department, such as WPI's, or the feature set does not include wireless transmission capability.

2.2.1.1 PDA Medic

PDA Medic is a product that is aimed directly at the Emergency Medical Services market. It offers the ability to enter patient and incident information for later synchronization to a database via the Palm OS's HotSync capability. This system also enables the synchronization of data via a modem or LAN connection. Additionally, the system provides the capability to store the information to a database for later access. However, this system costs roughly \$5,000 per department, so it is prohibitively expensive for WPI's EMS department.

2.2.1.2 MedDataSolutions

MedDataSolutions' Registrar product is aimed primarily at EMS technicians arriving at hospitals, who would normally need to fill out forms upon arrival. This package offers infrared printing capabilities and the ability to upload data through the Internet to its central database. Therefore, EMS departments delegate control of their data to MedDataSolutions and do not store it locally, thus giving up control of their data.

2.2.1.3 MobileEMS

MobileEMS is a standalone product designed to allow for field data collection by EMTs. It does not provide for any synchronization capability or database integration, nor does it provide for any wireless transmission of data.

2.2.1.4 RescueNet EMS Pro

Westech Mobile's RescueNet suite contains a variety of components, including dispatch, billing, transport requests, and patient information. This system would provide the needed functionality, but only by purchasing multiple components of the system and integrating them. This system is aimed at larger EMS organizations.

2.3 Medical Information Regulations

In the interests of protecting patient privacy, the federal government has developed guidelines for the handling of medical information. These standards are relevant to the project in regards to the manner in which information is entered, transmitted, and stored.

The Health Insurance Portability and Accountability Act of 1996 (also known as HIPAA) contained new provisions intended to protect medical information. Most of these standards apply to the transfer of data between entities; however, some of the standards may be applicable to the project in terms of protecting the output of the system.

There are multiple facets of the HIPAA that health care organizations have to address. The portions of the HIPAA regulations that affect this project are those relating to security. The regulations cited in this section come from the proposed rule from the Department of Health and Human Services (HHS). HHS has yet to issue the final rule with regard to the HIPAA security regulations, so no compliance deadline has yet been set. Once the rules are finalized and published in the Code of Federal Regulations (CFR), the rules will have a compliance deadline of two years from that date.

One regulatory term with particular significance to this project is “Protected Health Information” (PHI). PHI is “health information in any form (i.e., paper, electronic or verbal) that *personally identifies* a patient” (KIBB01a). A significant requirement of the HIPAA is that each party involved in using PHI can view only the “minimally necessary information” to perform the required tasks (KIBB01b). According to Kibbe, “The intent of the HIPAA standard is to discourage the current practice of open access to medical records that may contain large stores of information regarding a patient’s previous medical history.”

One facet of the HIPAA regulations is backup and recovery of medical information. The regulations require that health care providers create plans for data backup, disaster recovery, and emergency operations (STAN02). The regulations also require that virus protection software be installed on systems containing medical information.

The HIPAA regulations also provide technical requirements for systems containing PHI. Procedures for “emergency access” are required – these are described as “documented instructions for obtaining necessary information during a crisis” (STAN02). HIPAA regulations also require audit controls to monitor what accesses to the database have occurred. Additionally, they require that user-level or role-level access can be defined.

One facet of these security requirements is to verify that the data entered is valid data. According to the regulation, methods for satisfying this requirement include “check sum, double keying, a message authentication code, or digital signature” (STAN02). The HIPAA specifies requirements for the transmission of data. Any data sent over open networks is required to be encrypted, or have other access controls. Additionally, the regulation states that the communications must have the following features: (STAN02)

Integrity controls (a security mechanism employed to ensure the validity of the information being electronically transmitted or stored).

Message authentication (ensuring, typically with a message authentication code, that a message received (usually via a network) matches the message sent).

The HIPAA regulations are described as “technology neutral,” so the particular implementation details are left to the relevant health care organizations. Each regulation was considered in the development of this project.

2.4 Palm OS History and Overview

In designing Personal Digital Assistants, the goals were straightforward. The device was to be designed to be small and light enough to fit inside of a shirt pocket, have an ergonomic interface, allow for desktop integration, and last three weeks at a time on two AA batteries (FOST02). The first Palm PDA ever created, the Palm Pilot 1000, was developed in 1996. The Pilot 1000 had 128K of dynamic memory, a black and white screen, and no networking capability. The Pilot 5000 was soon to follow the 1000 with similar characteristics, but rather than 128K of dynamic memory, it had 512K. “The only link to the outside world was through the HotSync conduit.” (WINT01) Using a serial connection, the HotSync application synchronizes data on the handheld device with a desktop computer. Conduits are plug-ins for the HotSync Manager. These components exchange and synchronize the data (PALM03a).

In 1997, the Palm Pilot was introduced. This was the first Palm that had network connectivity. It had 1 MB of memory, which was enough for a network protocol stack. The new operating system also included a Net Library (NetLib), which gave developers access to the protocol stack. The Net Library was added to enable network connectivity via HotSync which allowed users to synchronize remotely with their desktops. Palm then introduced a wired modem to provide a dial-up Internet connection (WINT01).

A year later, the Palm III was developed, and was a large step in improving networking support. This device had 2MB of dynamic memory, which allowed for more resources to be left over after the protocol stack was loaded. These resources could be used to build networking applications. In the

same year, Palm introduced the Palm VII. This handheld device was equipped with a wireless radio, which enabled users to send and receive email, as well as browse the Web as long as there was coverage (WINT01).

The newest Palm operating system, released in February 2002, is Palm OS 5. It is equipped with the “Palm Application Compatibility Environment” (PACE), which allows developers an easier migration path to the new hardware. It is also accompanied by a new development tool, the Palm OS Simulator, which allows developers to test and debug applications on a desktop system that has a window with a “device” running Palm OS 5. The Simulator is an implementation of Palm OS 5 running natively as a Windows application. The new operating system also includes a strong set of standard Security Application Programming Interfaces (APIs) for 128-bit RC4, SHA1 and RSA-verify cryptography algorithms, as well as Secure Sockets Layer (SSL) 3.0/TLS 1.0 service (WINT01).

Applications running with PACE benefit from increases in speed and screens of high-density because the operating system and PACE are written in native Advanced RISC Machine (ARM) code. The ARM is a processor family that is essentially a Reduced Instruction Set Computer (RISC).

The biggest single change in Palm OS 5 is that it runs on a new hardware platform, using ARM®-compliant processors from industry leaders such as Intel, Motorola, and Texas Instruments. This allows dramatic improvements in speed and capability for the platform and the software that developers can create, while maintaining the low cost, power efficiency, simplicity of use, and form factor that have made Palm OS devices overwhelmingly popular. (PALM03b)

PACE also allows for Palm OS 4.0-compatible applications and data to be run on, and shared between, devices running the two separate operating systems. The first series of Palm OS 5-enabled devices on the market was the Sony Clie PEG-NX series.

It was not until Palm released version 3.2 of the Palm OS that handhelds had wireless connectivity. The introduction of the Palm VII was the first to have this capability. In older versions of the Palm OS, the TCP/IP communication standard was only available for Palms via a serial or modem connection. With the help of wireless cards, newer releases of the Palm OS have adopted the TCP/IP protocol as a standard in wireless connectivity.

2.5 Sony Clié PEG-NX60

The Sony Clié PEG-NX60 served as the development platform in this project. The handheld device is 5½ inches tall, 2-7/8 inches wide, 11/16 inches deep, and weighs approximately eight ounces. It features a 200 MHz processor, a “QWERTY” keyboard, a wireless communication slot, 16MB of RAM, and 16MB of ROM. The display resolution is 320 x 480 pixels. The nonstandard size is due to the combination of a 320 x 320 work area and an electronically displayed graffiti² area (which is typically not part of the display). This PDA includes two very important features: it runs Palm OS 5, and has a communication slot capable of supporting an 802.11b wireless Ethernet card.



Figure 1: The Sony Clié PEG-NX60 (SONY03)

The only shortcoming of the NX60 at this point is that no Ethernet Cradle has been designed for it. In the event WPI's wireless network goes down, there will be no way for technicians to upload data using a Clié series handheld

device until the network is running again. If another device were to be used that was compatible with an Ethernet cradle/cable, the data could be sent through a

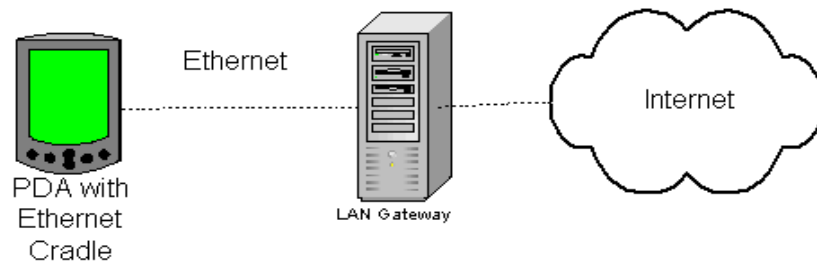


Figure 2: The Ethernet Cradle Connection Process (WINT01)

² Graffiti is a software system on the Palm's screen that converts a special type of shorthand into text. Used for inputting data on the handheld device.

“wired” connection to any Ethernet access point, much like most laptops have. See Figure 2: The Ethernet Cradle Connection Process. The following models are currently compatible with Ethernet Cradles:

Palm III Series

Palm VII Handheld

IBM WorkPad 20X and 30X Handhelds

Franklin Covey family of Handhelds (except the Palm V Series)

Symbol Technologies SPT 1500 Handhelds
(PALM03e)

Design is underway for Palm OS 5 devices to be compatible with Ethernet cradles/cables. When this effort is completed, most Sony Clié series models will be able to send and receive data through an Ethernet cable as well as a wireless connection. See Figure 1: The Sony Clié PEG-NX60, and section 9.2 for further images of the NX60.

2.5.1 PEGA – WL100

The PEGA – WL100 is the standard-issue wireless network card for the Clié series. This card provides access to the Internet through an access point which enables users to do things such as browse the Web, send email, and even synchronize data with their desktop PCs. The WL100 provides 128-bit WEP encryption for data that is transferred over the wireless medium. See Figure 3: The PEGA - WL100.



Figure 3: The PEGA - WL100
(SONY03)

2.6 Client-Side Data Entry

2.6.1 User Interface

Due to the size limitations of PDAs, designing an effective user interface (UI) is an important and challenging task because restrictions exist on the amount of information and controls that can be

displayed on the screen. Palm OS enabled devices commonly come in two resolutions, most often 160 x 160 pixels³ or 320 x 320 pixels.

Palm applications, also called thin clients, are made with forms that can include many user interface components, such as text fields, command buttons, and drop down menus. Forms may be larger than the display in the vertical direction, as vertical scrolling is a built in capability; however, the forms may not exceed the horizontal resolution.

Palm UIs should, above all, be simple and consistent with other Palm applications. Due to the low resolution, users will often determine a button's function through location and shape before reading the label. As Palms have been shown to produce eye strain after 10 minutes of prolonged use, a well designed interface is essential to eliminating unnecessary strain and to minimize time spent using the application (PALM03c).

2.6.2 Operating System

Current PDAs commonly come with one of two operating systems, either the Palm Operating System or Windows Compact Edition (CE). Currently, out of all PDA sales, there are about 17.5 million Palm OS users and 2 million Windows CE users (PALM03d). PDA operating systems are embedded, rather than installed as they would be on a personal computer, so changing or upgrading a PDA's operating system is difficult for the normal user.

Both operating systems offer a full developer's kit, online developer support, emulators, simulators, and comprehensive debugging tools. There is currently an open-source, command line developing platform for Palm OS, while retail software is the only way to program for the Windows CE environment. Palm applications are primarily developed in C, and Windows CE devices are programmed in a number of languages, including C, C++, and Visual Basic.

³ A Pixel is the basic unit of programmable color in a computer display or in a computer image (infoplease.com, 2002).

Windows CE offers a feature set that is currently unrivaled in the PDA world, including the native ability to create and edit Microsoft Office documents natively. Palm OS offers a smaller feature set but a much larger selection of programs. However, due to the complexity introduced to the PDA by the rich feature set, Windows CE has a much steeper learning curve than the Palm Operating System.

2.6.3 Palm Basics

Palm devices use a relatively unique approach to user interaction. The user may perform all input using a stylus, a small pointing device, to interact with the device's touch-sensitive screen. Text may be entered through a built in, or attached keyboard, but is most often entered through Graffiti®. Graffiti® is a proprietary method for entering characters through pre-defined strokes. Graffiti® operations may only be performed in the graffiti area, which is divided into a section for letters, and a section for numbers. Additionally, an on-screen keyboard is available for use. See Figure 4: The Graffiti Area.

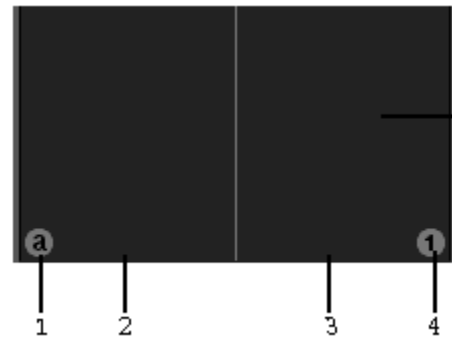
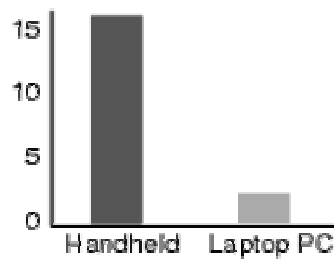


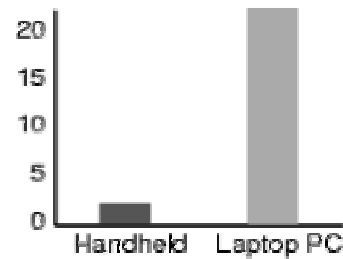
Figure 4: The Graffiti Area

1. Displays the on-screen Alpha-Keyboard
2. Alpha Text Area
3. Numeric Text Area
4. Displays on-screen Numeric Keyboard

The Palm OS is designed to feel like one large application, with each program designed as a feature set. As such, the user never explicitly exits a program, but simply accesses another “feature.” A highly consistent UI enables this



Accesses per day



Average session time (minutes)

Source: Palm, Inc. user surveys.

Figure 5: Palm access frequency

illusion to exist; an inconsistent UI may lead to user dissatisfaction and confusion. See Figure 5: Palm access frequency.

The Palm OS is single-threaded, and its applications must wait for one thread to complete before starting another. This makes it essential for operations to be extremely fast, or for a dialog to inform the user of status. This has particular implications for network access, as the user must wait for all network operations to be finished before continuing.

2.6.4 Palm Applications

Each application on a Palm handheld device is essentially a database. There are two specific types of databases, PDB and PRC. Both are “record databases that contain records used with applications ... Palm resource databases contain application resources, including the code and the user interface objects for the application. These databases are stored in memory on handheld devices, and are stored in file form on desktop computers.” (PALM03h)

Each type of database contains the following components: (PALM03h)

- A header containing fields that describe the database and refer to the information blocks and raw record data in the database.

- A list of record entries, each of which describes a block of raw record or resource data.

- Two optional information blocks: the application information block and the sort information block.

- The raw record data, which is stored in linear format and referenced from the record list in the header.

The Emergency Incident Reporting Application is a .PRC file. It contains an Application Info Block, which contains arbitrary data, and Raw Data.

2.6.4.1 Application Development

The Palm Operating System natively supports only one language, and that language is C. The Palm OS can be extended to use several types of languages, but these must be further interpreted and can cause a needless reduction in efficiency. The most common development tool used for writing Palm OS applications is Metrowerks CodeWarrior. The CodeWarrior Integrated Development

Environment (IDE) contains tools for writing, compiling, linking and debugging source code. The Constructor for the Palm OS, which is also included in the CodeWarrior Package, is a resource editor with a graphical interface. These resources are combined with the source code to create the final program.

2.6.4.2 Application Implementation

According to the Palm OS User Interface Guidelines:

The Palm handheld is fast to use and easy to learn. Because the handheld's main purpose is to make it easy to organize and manage your life, it requires a minimal learning curve. Users must be able to pick up a Palm Powered handheld and, with no training or instruction, navigate between applications (without getting stuck) and execute basic commands within five minutes. (PALM03c)

This has several implications for the client-side application, but the biggest is the impact of perceived speed. Due to the number and frequency of accesses on a PDA, users do not want to wait for a command to execute. Optimizing an application for this kind of use is important for a satisfied user. Figure 5 illustrates common user access profiles for Palms versus PCs.

Another key goal in application development on the Palm Operating System is the minimization of "taps" (selections with the stylus). An application should be optimized to require the least amount of input possible to get to any possible point in the application. This means simple navigation for common components, and utilization of the drop down menu system for less frequently accessed feature sets.

Developing for the Palm OS requires a shift in attitudes, designing in terms of applications that would work well on the PC will not go over well on a PDA system. A PDA is seen as an extension of the desktop PC (PALM03c).

2.6.5 Formatting Data for Transmission

The ideal language for formatting data for this application was found to be the Extensible Markup Language (XML). XML, a dialect of the Standard Generalized Markup Language (SGML), allows the

creation of descriptive data documents that can be read on any platform. The use of a non-platform-specific data format is important to eliminate dependence on a specific back-end operating system or hardware configuration. The use of XML over a plain text format improves the flexibility, ease of development, and efficiency of back-end processing of the transmitted information.

2.7 Data Transmission

The ultimate goal of this project is to enable the EMS technicians to transmit an electronic copy of their incident report from the PDA to a database that exists on a desktop computer. Transmission of the data from the handheld to the desktop computer was the second major phase of this project. In dealing with PDA's, there are two customary ways in which the data can be transmitted: with a serial connection, or via a wireless network.

2.7.1 Direct Serial Access

Data can be transmitted from the PDA to a desktop computer via a serial connection, known as Direct Serial Access. The PDA is placed in a cradle that is connected to the desktop computer. At this point, the HotSync

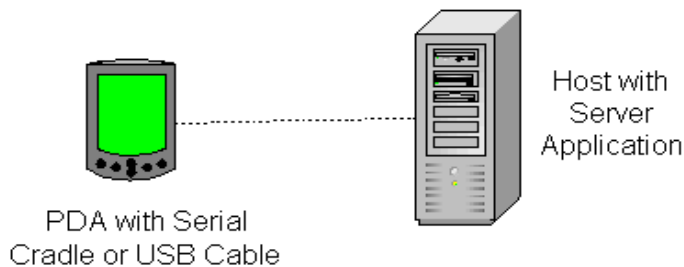


Figure 6: Direct Serial Access Connectivity
(SONY03)

manager can run conduits that synchronize data between the PDA and desktop machine. See Figure 6: Direct Serial Access Connectivity.

2.7.2 Wireless Transmission

Worcester Polytechnic Institute's wireless network is based on the 802.11b wireless transmission standard, alternatively known as Wi-Fi, which extends WPI's wired LAN. It is a Radio Frequency (RF) wireless network, which is optimal for networking computers over short and medium-range distances. These distances can range up to several hundred feet; depending upon whether or not the signal is being transmitted from an indoor or outdoor location. Wi-Fi is a non-directional technology,

which means it does not require a line-of-sight to make a connection. See section Figure 7: Wireless Connection Process of a PDA.

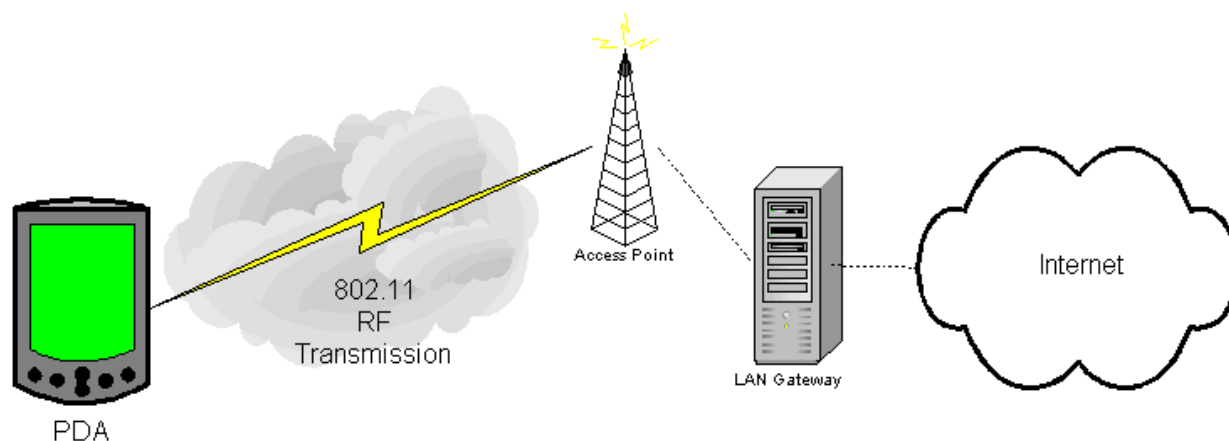


Figure 7: Wireless Connection Process of a PDA
(WINT01)

802.11b, a spread spectrum⁴ technique known as HR-DSSS (High Rate Direct Sequence Spread Spectrum) is now commonplace among most office buildings, airports and campuses, and continues to grow at a rapid pace. It uses 11 million chips per second to achieve 11 Mbps (megabits per second) in a 2.4-Ghz (Gigahertz) band. “The data rate may be dynamically adapted during operation to achieve the optimum speed possible under current conditions of load and noise. In practice, the operating speed of 802.11b is nearly always 11Mbps.” (TANE03)

A Wi-Fi connection is usually made from a PDA or laptop to a wireless Local Area Network (LAN) access point. “The access point connects to your network router/hub and forms what can be thought of as an invisible Ethernet connection between the hub and any Wi-Fi-enabled computer in range” (BACH02). Wi-Fi technology also allows for data transmissions to be sent at rates of multiple megabytes per second. In the case of an RF wireless network, the device is limited only by the speed of that network.

⁴ A Spread Spectrum uses wide band, noise-like signals that are much wider than normal signals, and therefore, are harder to detect and intercept.

When dealing with wireless technology, coverage is a major issue. This possible downside is dependent on location, and wireless networks in general. Whether or not

Table 1: WPI Wireless Access Points

<p>Gordon Library - Basement, 1st, 2nd, and 3rd Floors. Campus Center - All floors as well as the front and back patios. Freeman Plaza / Reunion Plaza (Fountain) area of West Street. Olin Hall 107 Classroom. Higgins Labs - 1st, 2nd, and 3rd Floors including the Discovery Classroom. Fuller Labs - Basement, 1st, 2nd, and 3rd Floors including Perreault Auditorium. Atwater Kent - 1st, 2nd, and 3rd Floors including Newell Hall. Alden Hall - Great Hall. Harrington Auditorium - Basketball court and stands. Morgan Hall - Dining Hall. Founders Hall – Dining Hall.</p>
--

this is an issue in this instance will be dependent on how mobile the EMS technicians need to be. At this point, WPI's wireless network extends access points listed in Table 1: WPI Wireless Access

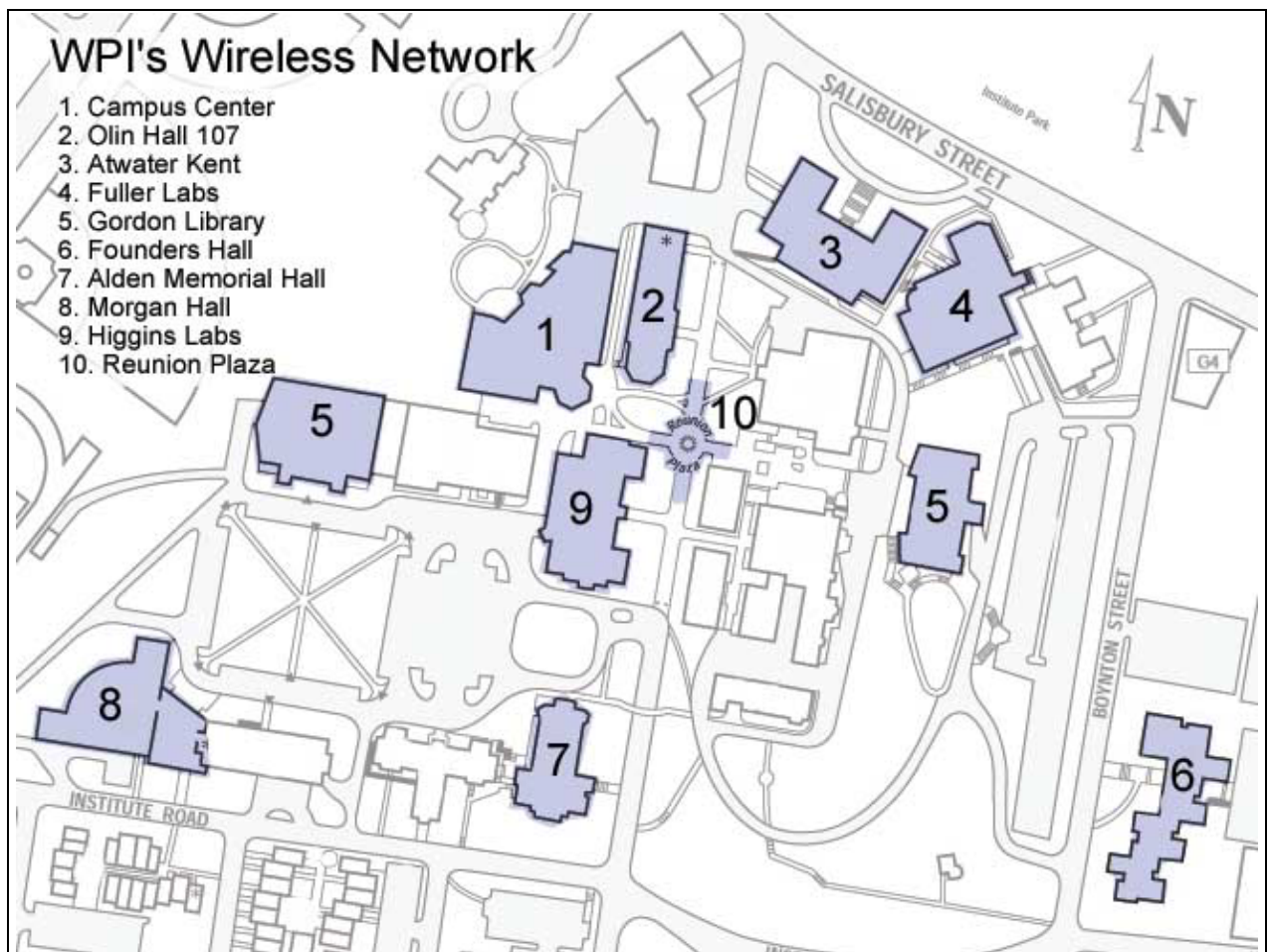


Figure 8 WPI's Wireless Network

Points. Since EMS technicians only respond to incidents on campus, these sites will be easily accessible, and will provide the necessary coverage the technicians need for uploading their incident reports. See Table 1: WPI Wireless Access Points.

2.7.2.1 Network Protocol Stack

When the application sends the data, the data travels through each layer of the network protocol stack. The network protocol stack prepares the data to be sent across the network. It is not concerned with what the data is; rather just that it reaches its destination. Each layer of the stack adds a protocol-specific header to the packet of data as it is sent, and strips off each header when a packet is received. The new packet that is created is then passed on to the next layer below it. See

Figure 9: The Network Protocol Stack (WINT01).

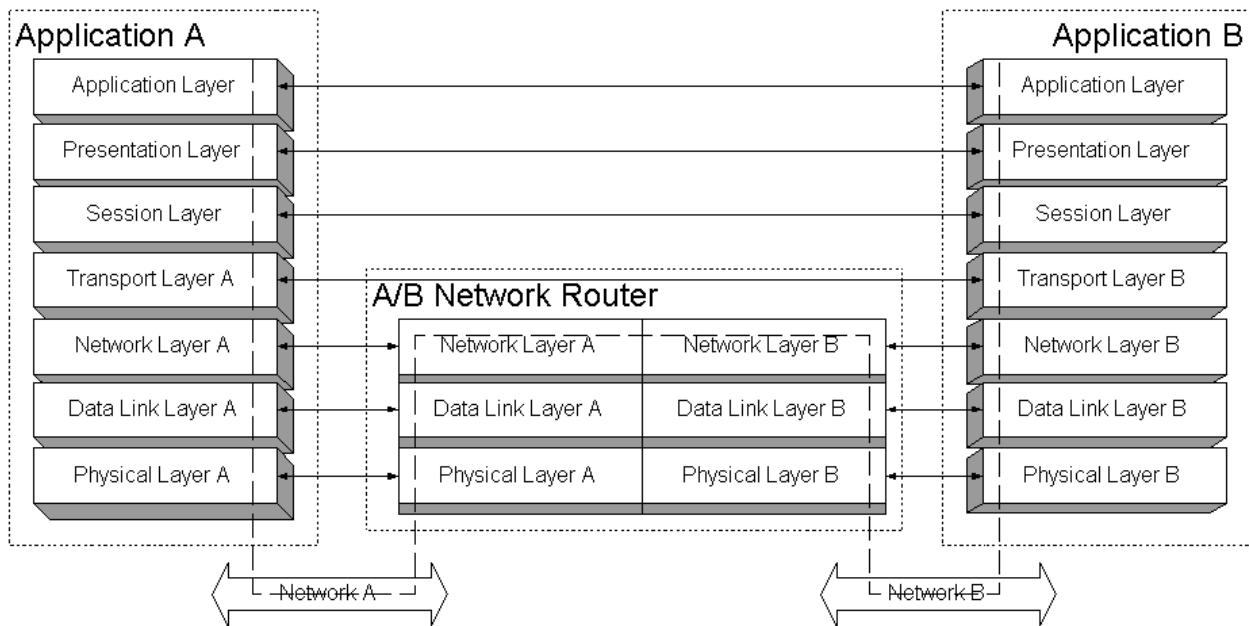


Figure 9: The Network Protocol Stack (WINT01)

The transport layer resides at the top of the stack. It is responsible for the overall end-to-end trip of the packet of data. If the amount of data being sent exceeds the transport's maximum packet size, the transport layer may break the data up into multiple packets. A transport packet is created when

the layer adds a transport header to the existing data packet, which is then passed on to the network layer.

The network layer routes the data received from the transport layer to a remote host. If the packet of data received by the network layer is fragmented due to its passage through the network, the network layer reassembles the packet. This layer adds a network header to the existing data packet and sends it to the data link layer.

The data link layer adds a media-specific header to the packet of data it receives. It passes the data to the physical layer, or network adapter. The network adapter writes the packet to the network medium as a stream of bits. In the case of wireless technology, this network medium would be in the form of radio signals.

At the remote host, this process is reversed. The network adapter translates bit streams on the network medium into data. This data is passed up the network protocol stack. As the packet travels up the protocol stack, each layer removes the header information added to the packet by its corresponding protocol on the server. If the data is fragmented, it is reassembled. (WINT01)

2.7.2.2 TCP/IP

Originally developed by a community of researchers centered on the ARPAnet, the Transmission Control Protocol and Internet Protocol suite (TCP/IP) has become a standard in data transmission between computers that share resources across a network. TCP and IP are the two protocols that make up the “Internet protocol suite.” TCP is responsible for verifying the correct delivery of data from a client to a server in a network connection. A server is a system that provides a specific service to the network. A process on a server listens to a port to get incoming requests from a client. A client is another system that uses that service, and sends requests to a specific port. TCP is a connection-oriented transport protocol. That is, “...they acknowledge received packets, resend them if they are not acknowledged, and reassemble the packets as they were originally sent” (WINT01). TCP keeps track of what information is sent, and in the event there was data that was lost, TCP can re-transmit the data. If any message is too large for one datagram (a collection of data to be sent; one unit of data), TCP has the capability of separating the message into several datagrams, and make sure that

they all get sent properly. The TCP protocol calls on the services of IP. The Internet Protocol is responsible for routing individual datagrams. Applications using the TCP/IP protocol consist of 4 layers: (HEDR87)

- an application protocol

- a TCP protocol that provides services needed by many applications

- IP, which provides the basic service of deciding what routes datagrams take to their destination

- the data link protocols needed to manage a specific physical medium, such as Ethernet or a point to point line

In today's wireless world, TCP/IP protocols are the optimal solution for handheld device network development. In the event that there was no acknowledgement from the server, TCP's ability to retransmit data is ideal for wireless communication. This is especially ideal in the case of this project. There may be instances where EMS technicians are sending data and go out of range while sending. This is a common problem with the wireless connectivity of handheld devices, so if there is no acknowledgement from the server, TCP will wait until another connection is established, and retransmit the data.

2.7.2.3 Packet Formulation

Using the TCP/IP protocol, data is sent in the form of datagrams, or packets. A packet has a physical appearance on an Ethernet or wire in a network. In most cases, a packet simply contains a datagram, which is a specific type of packet, so there is very little difference between the two, and a distinction between the two terms is negligible. TCP treats data as a stream of bytes. It assigns a sequence number to each byte. The TCP packet has a header that details which byte the packet starts with, and how many bytes of data it contains. The receiver can detect missing or incorrectly sequenced packets. TCP acknowledges data that has been received, and in the event of an error, retransmits data that has been lost along the network.

In creating datagrams to be sent via TCP/IP, headers are attached to the original packet by each layer. These headers are nothing more than a few additional octets added onto the beginning of a

datagram to keep track of it. TCP knows the maximum size of a datagram that the network can handle and manages the stream of data into separate pieces accordingly. A checksum is computed by adding up all the octets in the datagram. This result is placed into the header of the datagram. The TCP receiver computes the checksum again, and if they are not the same, the datagram is deleted because something went wrong with the transmission.

2.7.2.4 Sensing a Connection

Ultimately, a connection is a link between two processes, either on the same machine or two machines attached to a network. In networking, IP addresses are required to establish a connection. IP addresses are 32-bit integers that uniquely identify a host machine on a network. The first 16 bits specify the Network ID, and the second 16 bits specify the Host ID. Along with an IP address of the host, a specific port is also needed for a connection to be established. Finally, sockets are also used to help establish a connection.

Ethernet has its own addresses. This is to ensure that no two machines end up with the same Ethernet address, and to not place the burden on the user with having to worry about assigning addresses. Each Ethernet controller comes with a built-in address. Ethernet addresses consist of 48 bits. When a packet is sent out on an Ethernet, that packet is visible to every machine on that network. Ethernet headers are in place to ensure the packet is sent to the correct machine. Every machine has a table that links an Ethernet address to an Internet address.

To obtain the IP address of a destination machine based on its hostname, clients on a network send requests to a name resolution server to perform the name-to-address translation using the Domain Naming System (DNS) protocol. The protocol stack passes the name resolution request to this server. This server then reads the rightmost domain of the hostname and locates this domain on the Internet. Winton describes the DNS resolution process:

This domain then reads the next rightmost name and checks if it represents a host or sub domain. If it is a host, it gets the host's IP address and sends it as a response to the original DNS server. If it is a sub domain, it forwards the request to that sub domain for resolution. This process continues until either a host is located or a name cannot be found. If the host is

found, the response is sent back along the same route of DNS servers that the request traveled until it reaches the host that made the initial request. This entire process is hidden from the network application by the protocol stack and sockets API, which is the Net Library API. The API provides function that retrieve host info by hostname and IP address. Once the application finds a remote host, it still needs to locate a specific service to access on that host. The Palm OS Net Library supports service resolution only as far as retrieving the service information by name. (WINT01)

Regardless of how the handheld device is connected to the network, all applications on the device will access the network the same way, via the Net Library. Once the client has sensed a connection, a socket needs to be created. A socket is an end-point on a connection between two processes on separate hosts. A socket must be opened first, which creates an instance of the socket and allocates the resources of that socket. The Palm OS Net Library provides the Sockets API which provides a Sockets interface to the developer as a system library. Sockets lie between the application and the protocol stack and minimize the dependence that the application has on its underlying protocol stack. “The Sockets interface provides an abstraction of receiving data from or sending it to a remote host that allows the Network Application Developer to treat this connection as just another stream of information. The Sockets API emulates the Unix file I/O API that it was derived from.” (WINT01)

Connections in the Palm OS adhere to the general sockets model. However, there is one limitation. There are only four sockets that are available for all processes running on a device. “Palm OS devices are meant to perform only one specific networking task at a time.” (WINT01) To connect to a socket, the application opens a socket and makes sure there is network access. The remote host will, at that point, either accept or reject the connection. The lifetime of a socket can be summarized into five simple steps. First, the socket is opened. The socket then connects. Data is exchanged between hosts. The socket is disconnected, and finally, the socket is closed.

2.7.3 Security

Security is a major issue when sending medical information over a network. Security becomes even more of an issue when dealing with information sent over a wireless network. Since the data is being sent through the air, it is susceptible to anyone within range if there is no security mechanism provided to protect it.

An 802.11 wireless network provides two different methods of ensuring security: *authentication* and *encryption*. Authentication is the means by which a machine is granted authorization to communicate with another machine in a given area of coverage. There are two types of authentication: *Open System* or *Shared Key*. In an Open System, any machine may request authentication. The machine receiving the request can allow authentication to any request, or only those from stations on a user-defined list. In a Shared Key system, only stations which possess a secret encrypted key can be authenticated. Shared Key authentication is available only to systems having the optional encryption capability.

For the purposes of this project, the encryption technique was used to provide security for the medical information that is sent across WPI's wireless medium. Encryption is intended to provide a level of security similar to that of a wired LAN. There are two layers of encryption in regards to wireless transmissions: an encryption standard devoted entirely to 802.11b wireless networks, and Secure Sockets Layer encryption.

2.7.3.1 Wired Equivalent Privacy (WEP)

The Wired Equivalent Privacy (WEP) feature, which is part of the 802.11 standard, uses the RC4 PRNG algorithm from RSA Data Security, Inc. The WEP algorithm was selected because it is strong, self-synchronizing, and exportable (GEIE02).

The earliest 802.11b implementations provided 40-bit WEP, which was generally regarded as too weak to afford any real protection. Later 802.11b products (like the ones on the market today) strengthened WEP to use 64-bit (which is actually the same as 40-bit) or 128-bit keys. 802.11a products offer those same WEP levels but add a yet higher level--152-bit, while the some of the latest 802.11b+ products often feature 256-bit WEP. (MORA02)

If WEP is activated, the Network Interface Card (NIC) encrypts the payload of each frame before transmission using the RC4 algorithm mentioned above. The access point acts as a receiver and decrypts the frame when it arrives. A data stream encrypted with WEP can still be intercepted, but the encryption of the data makes it so that it is impossible to understand by the interceptor. When a frame enters the wired side of the network, WEP no longer applies since it only encrypts data

between 802.11 stations (GEIE02). The principle behind WEP is similar to that used by Secure Sockets Layer (SSL) which encrypts data sent between two machines over a wired network.

2.7.3.1.1 *Media Access Control (MAC)*

Every 802.11 device has its own particular Media Access Control (MAC) address hard-coded into it. The MAC layer is a sub-layer of the data link layer. This unique identifier can be used to provide security for wireless networks. When a network uses a MAC table, only the 802.11 radios that have had their MAC addresses added to that network's MAC table will be able to get onto the network (MORA02).

Like all networking devices, the wireless card used in this project has a unique MAC address encoded into its memory. Wi-Fi routers and access points that support a technique known as MAC filtering allow the developer to specify a list of MAC addresses that may connect to the access point using the MAC Access Control List (ACL), which limits the wireless connection to only those Wi-Fi radios whose MAC addresses are directly permitted to connect to the access point. “This method is similar to blocking a telephone call” (MORA02). If a foreign wireless radio with a MAC address that is not in the table tries to establish a connection, it will be rejected by the access point (WIFI02). This feature that deals with the issue of unauthorized access is known as MAC filtering (MORA02).

2.7.3.2 Secure Sockets Layer (SSL)

Digital certificates encrypt data using Secure Sockets Layer (SSL) technology. SSL is an industry-standard method for protecting Web-based communications that was developed by Netscape Communications Corporation. The SSL security protocol provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. There are two common strengths of SSL: 40-bit and 128-bit. Strength refers to the length of the “session key” generated by every encrypted transaction. The longer the session key, the more difficult it is to break the encryption. The Palm API has an SSL Library (SSLlib) that rests on top of the Network Library. SSLlib is utilized to provide SSL security to NetLib function calls which send and receive

data, **NetLibSend** and **NetLibReceive** respectively. SSLLib is Palm OS 5-specific. If it were to be implemented in the system, it would restrict EMS to using only devices that run the Palm 5 Operating System.

2.8 Server-side Processing

2.8.1 Data Processing Server

This project requires a server process to be running at all times to receive the data transmitted by the PDA client software through the network. Once the server has received the data from the handheld client, it translates the data into a database-loadable format and loads it into a database.

2.8.1.1 Data Translation

Because of its data-centric nature, XML is the data format that was chosen for transmitting data from a handheld device to the server. However, XML does not have a relational design and XML documents must be translated to be useful to a relational database. As Server Query Language (SQL) is the predominant relational database manipulation language, any successful translation would need to convert XML to SQL. A variety of methods exists for this translation; however, most of the existing tools for accomplishing this are database management system (DBMS)-specific.

Oracle has embraced the XML standard by completely integrating it into Release 2 of its Oracle 9i DBMS, dubbed "XML DB." As such, it can retrieve data from both XML documents and relations seamlessly (VOTS02). However, the usefulness of this capability to this application is limited; there is no need to *store* the data in XML form, only to transmit it in that format.

Oracle's 8i database system approached the XML problem from a middleware approach, as opposed to 9i's integrated approach. In 8i, applications and server components exist to convert XML data to SQL queries, and vice versa (MUEN02). This approach, on its face, appears to more closely mirror the needs of the project. However, it would seem that 9i would offer all of the capabilities of the 8i DBMS product.

One simple method for loading XML data into Oracle databases, beginning with its middleware approach in 8i, is to convert the hierarchical XML document into a row-column type structure. This is commonly done using Extensible Style sheet Language for Transformations (XSLT). By creating an XSLT document, XML data can be converted into a row-column structure and loaded into multiple tables in a relational database (MUEN02).

The Oracle 8i DBMS also offers the ability to store data based on the object-relational model, as opposed to the simple relational model that is most often found in database management systems. For instance, object types can be defined and used hierarchically, similar to the paradigm of object-oriented programming. This data model fits very cleanly with the hierarchical data model most often found in XML documents (BANE00). Thus, it is likely that little to no XSLT transformation would be required to import data from XML into an object-relational database.

2.8.2 Database Implementation

An important component of the project is to store the information that is entered into the handheld application and sent to the server. Previously, EMS used a Microsoft Access database to store the information that is entered by hand from their run sheets. This information was entered monthly, from run sheets that were stored until entered into the database.

2.8.2.1 Database Design

The existing database used by WPI EMS had a relational design consisting of three tables: call information, call type, and member information. The existing database did not fully account for responder information and tie it to the incident records, so that was a necessary addition to the new system. Medical privacy requirements dictated special considerations in the database design as well. For instance, searching for incidents by patient name is prohibited by law.

2.8.2.1.1 *Relational*

The relational database model is currently the most mature and common data model. In the relational model, data is organized into relations (also called tables). Each piece of information about an item is called an attribute (column). Each instance of an item is stored as a tuple (row) in the relation. Relations are related to each other using primary keys and foreign keys stored in attributes of a tuple (ULLM97).

2.8.2.1.2 *Object-relational*

The object-relational model is a relatively new data model that combines the object-oriented programming approach with the relational data model. Abstract data types (ADTs) can be defined, and relations are defined as instances of those types. These ADTs are included in the SQL3 specification. One aspect of pure object-orientation (OO) that is missing in ADTs is the “encapsulation” provided by the public methods in OO programming. ADTs may be accessed by any valid SQL3 queries, and their access is not restricted by predefined methods (ULLM97).

2.8.2.2 DBMS Selection

To facilitate the transfer of data into a database using the methods described in section 2.8.1.1, it was necessary to select a new database management system. Factors such as platform, scalability, XML translation capability, and cost were considered in this selection. Additionally, reporting capabilities are important to the customer, although that decision may be isolated from DBMS selection.

Due to the sensitive nature of the data being stored in the database, each DBMS’s security capabilities were important. The security features provided by the selected DBMS influenced the project’s data design.

2.8.2.2.1 *Microsoft Access*

Microsoft Access is a relatively simple and small-scale database system. Its primary focus is on desktop users, and thus, contains built-in forms and reporting features. It is file-based, and supports the Open Database Connectivity (ODBC) standard, as well as the Microsoft-specific Object Linking and Embedding Database (OLEDB) connectivity methods. Additionally, it supports the relational data model and SQL. Access databases can only be directly accessed on Microsoft Windows platforms. However, they may be accessed using SQL over ODBC connections from any ODBC-capable OS.

Microsoft Access's security permits object-level permission assignment. For instance, each table may be assigned a different set of permissions for each user. Additionally, there are workarounds in Access that permit "assignment" of permissions at the tuple or attribute level. To do this, permissions on a source table would be set at a restrictive level. If the database administrator wanted to grant access to only certain attributes of a table, for instance, they could create a new query that only returned results from those attributes. They would then grant users permissions to the new query only. The query would run "as" the user with the permissions to the original table (CHIP00).

2.8.2.2.2 *Oracle*

The Oracle Corporation develops a very widely-used Relational Database Management System (RDBMS) that supports a variety of advanced features and platforms. The main connection method used to connect to Oracle databases is Net8, or Oracle Net, which is capable of running over TCP/IP connections (GREE01). Oracle databases also support the ODBC connectivity standard, as well as the Java Database Connectivity (JDBC) standard. Oracle offers its products for free download at its site for development purposes only. Production licenses start at \$400 for the Personal Edition.

Like Access, Oracle provides table-level and view-level permission granting. The same type of workaround as described in the Access section would be used to grant access only to certain

attributes of a table, using views instead of queries. Oracle provides modules to encrypt and decrypt select data before storing it in the database. This allows for an extra layer of security to prevent unauthorized access to data at the server level.

2.8.2.2.3 *MySQL*

MySQL is an open source RDBMS that, like Oracle, runs on a variety of platforms. MySQL was originally created to fill a void in small-range to mid-range SQL-driven RDBMSes. At the time of its creation in the mid 1990s, the only SQL-driven relational database management systems were those created by companies such as Oracle, IBM, and Informix. These RDBMSes required very large monetary and technology investments (KING99). MySQL supports the ODBC connectivity standard, as well as the JDBC toolkit for Java applications. MySQL supports table-level and attribute-level permission specification. MySQL provides the ability to directly control column read permissions, such that no workaround is required to grant some users permission to a subset of a table's attributes.

2.8.2.2.4 *SQL Server*

Microsoft SQL Server is an RDBMS aimed at the midrange relational database server market. Microsoft Access provides a mechanism to automatically “upsized” a database for entry into SQL Server, when additional data capacity and features are needed. SQL Server provides built-in utilities for loading XML documents into the relational database, as well as storing XML documents in the database and accessing them in their native format.

SQL Server only runs on the Windows platform, but, using its ODBC and JDBC support, can be accessed from other platforms. One particularly important point about SQL Server is its price: \$1,489 for a server license with five Client Access Licenses. SQL Server, like MySQL, supports attribute-level permissions without a view workaround.

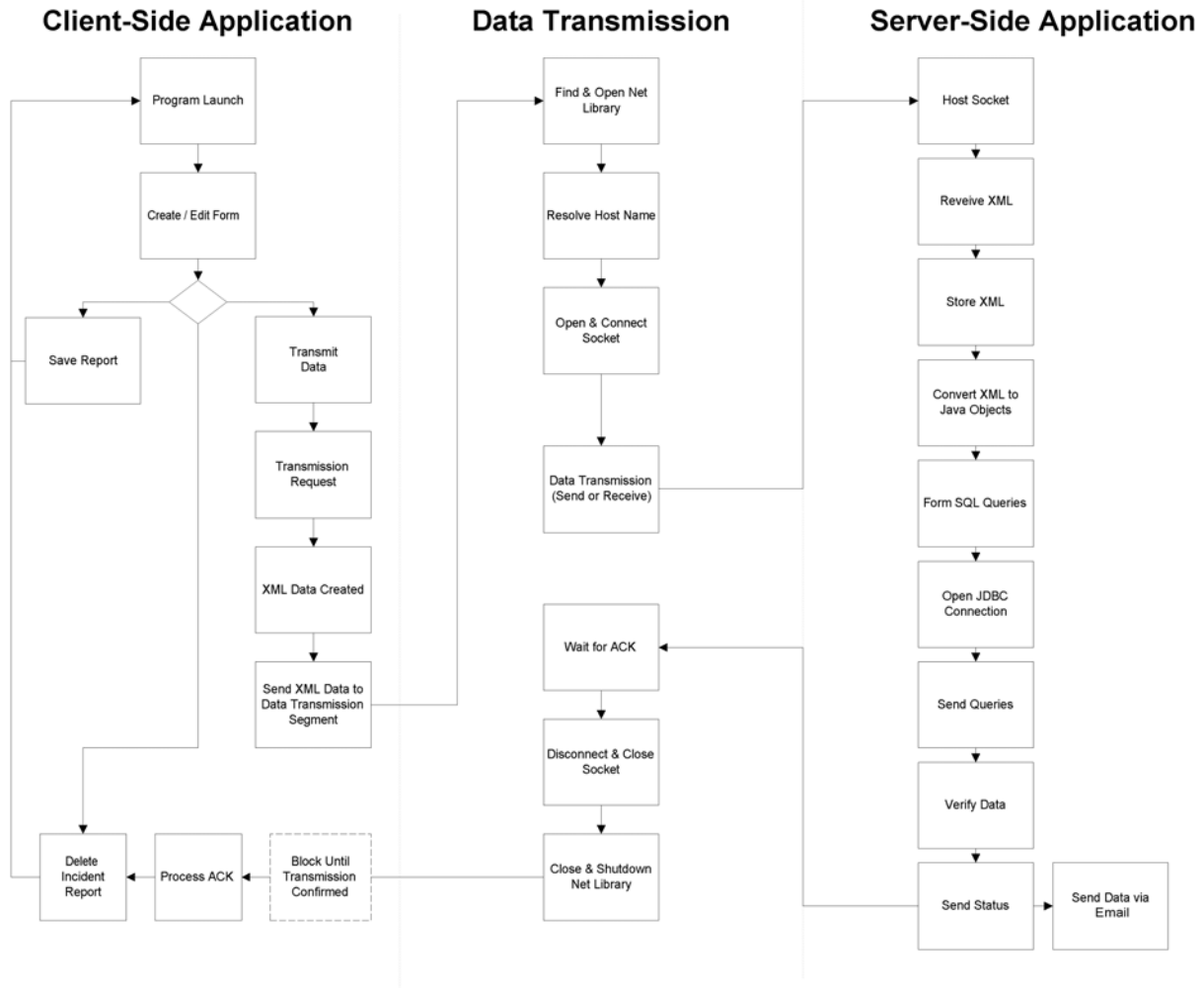


Figure 10: High Level System Flow

3 Methodology

3.1 Introduction

Based on the information obtained from the literature review, certain technologies were chosen with which to implement the system with an eye toward the WPI EMS group’s requirements. The methods and technologies used are discussed in the next section.

A high-level approach to the system is quite simple to understand. A client application is created for the Palm OS 5 operating system and resides on the handheld device. This application is responsible for collecting user input through the various forms it presents to its user. It is then responsible for storing the input from the user in a database that is created on the device. The user has the option of

transmitting each incident report (record in the database) to the EMS Server via two methods. One method is by way of direct serial access (HotSync); the other is by way of wireless transmission. Once the data is received by the server, it is then parsed and inserted into the EMS database. The overall design breaks down into three major components, the Client-side Application, Data Transmission component, and the Server-side processing component. Each component of the system is comprised of their own sub-components with specific logic and code that is discussed in the following sections. See Figure 10: High Level System Flow for a flowchart detailing the flow of data within the system..

3.2 Client-Side Application

3.2.1 User-Friendliness

For the application to improve the efficiency of the WPI EMS group, it needed to improve on the incident recording system as it existed previously. The efficiency and ease of use of the data-entry system is essential to the success and acceptance of this program with the EMS corps. Careful consideration was given to how the EMS technicians will use the device itself, as well as the client-side application. The design decisions focused on ensuring that the maximum ease of usability is achieved on the part of each technician. One goal of the system was that a technician's ability to file a report should not be hindered by either the device or the application.

3.2.1.1 Choosing the Device

The device selected for this project was done so because it supported all the needs of the EMS technician. Since the EMS technicians will be carrying the device around with them while they are on duty, they need a device that is both lightweight and durable. The Sony Clié PEG-NX60 offers both of these features. It is extremely durable for a handheld device, and weights only 8 ounces (220g). If, for some reason, things become rough for the technician in the field, the NX60's durable structure will protect it from most common accidents that could occur.

A number of devices could have carried out the functionality of this project. However, the NX60 comes with a few key parts that give a great deal of usability to the user. It is equipped with a

“QWERTY” style keyboard, which offers a simple data entry solution for the novice and occasional user, as well as the proprietary Graffiti system, which has a steeper learning curve but improves overall data entry speed. It also has a navigation scroll pad on its left side which allows the user to scroll through different parts of an application without having to touch the stylus to the screen. While holding the device in their left hand, the user can easily scroll and use the stylus for other interactions with the screen at the same time, such as toggling between applications and selecting parts of an application. The NX60 also has a built-in voice recording unit. This may be advantageous to the technician if they need to record a message needed later to remind them of something that went on at an accident scene. This is a much faster way of taking notes since there would be no need to have to write them to the device.

In addition to the sleek usability enhancements provided to the user, the NX60 is equipped with a wireless communication slot, which allows for an 802.11b wireless card to be installed enabling wireless LAN network capability. The ability for the device to be able to connect to a wireless LAN was an essential part of the requirements for determining which device to choose for this project. As discussed in section 2.5.1, the WL100 wireless card is the standard issue wireless card for the Clie series handheld, and was the card chosen to enable wireless network access in this project.

3.2.1.2 Choosing the Operating System

While both the Palm OS and Windows CE operating systems are capable of performing the task set forward by this project, due to the wide availability, ease of development, and low learning curve, the Palm Operating System was chosen. Furthermore, the NX60 supports only the Palm OS 5 operating system. Palm OS 5 also supports SSL, which allows for encryption of the data being sent over the wireless medium.

3.2.1.3 Designing the User Interface

The user interface followed the design suggestions laid out by the Palm Corporation (PALM03f). Further interface design was based on a user analysis of the EMTs, and other design requirements laid out by the EMS group.

3.2.2 Emergency Incident Reporting Design

The client side application, known as the Emergency Incident Reporting (EIR) system, was developed with two primary goals in mind; speed and ease of use. With this goal in mind, the application was designed to be sleek and self explanatory. Few choices should ever presented to the user at once, allowing for quick and easy decision making while in the field, hopefully eliminating any chance of the program becoming a burden to the technician in the field. One of the primary obstacles faced in the design process was making a robust application that fulfilled these needs.

The data entry system follows as closely as possible the format set forth by the EMS' former system, while some changes were necessary to make the conversion from paper to PDA a successful one. The data fields set forth in the application mirror the data fields present in the paper system, and improve on that system by automating various data entry tasks, such as date entry and the recording of on site personnel.

Emergency Incident Reporting: Client Side High Level Diagram

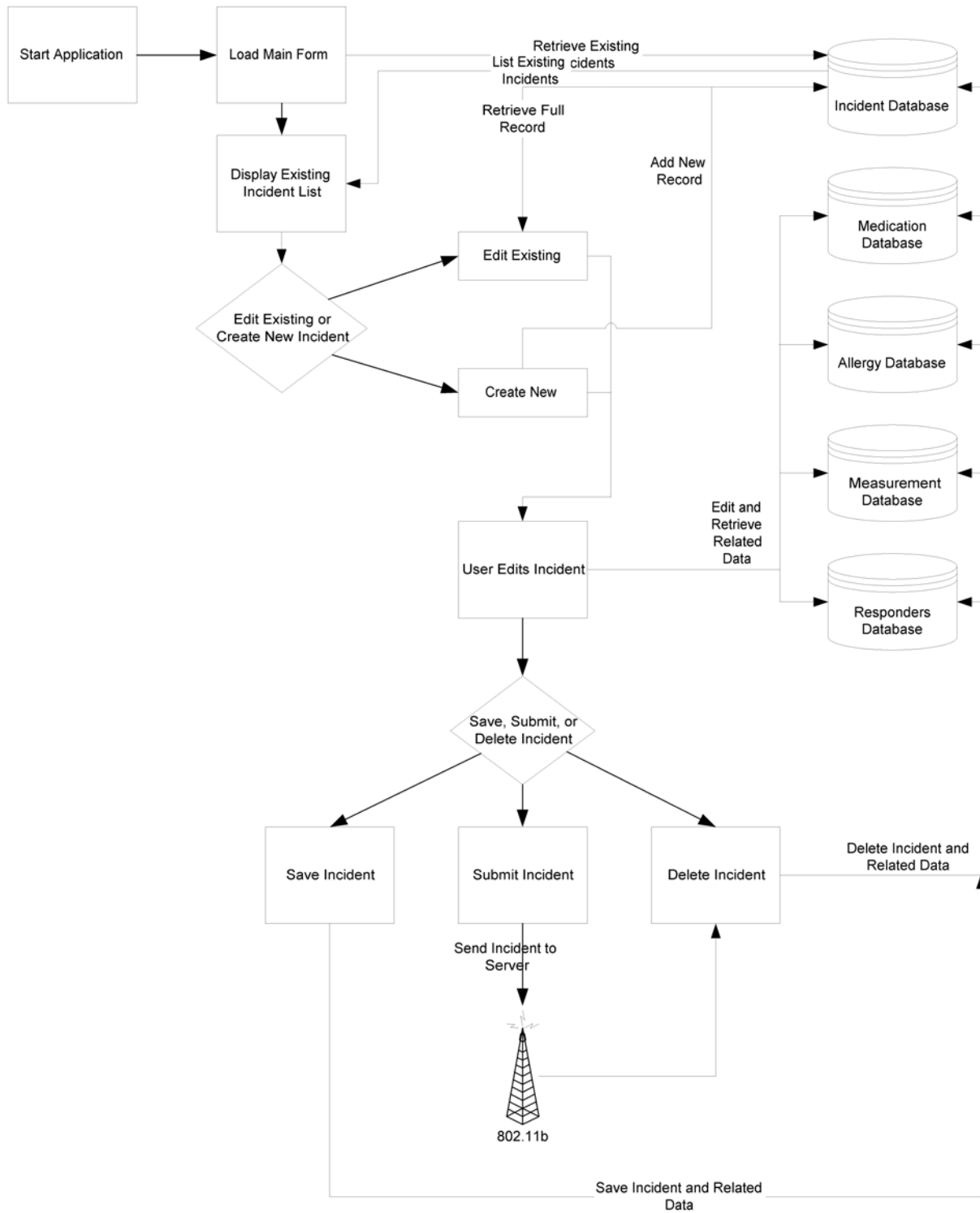


Figure 11: High Level Client Side Diagram

3.2.2.1 Client-Side Application Narrative

To use the application, the user selects the icon representing the EMS program from the main menu on the device. The file launches to a screen listing previous un-transmitted entries, as well as two buttons, a “New” button, and a “Submit” button.

Upon selecting “New” (which creates a new incident report), the user is taken step-by-step, through all the necessary forms, filling them out or skipping them as they wish. If the user selects a previous entry (by tapping on it) they will be brought to a screen with data populated by the previous incident. At any point during either process, the user may decide to save the form, send it to the server, or delete it. If they save the form, it saves that incident report and returns to the list on the main form. If they decided to send it, it is sent to the server and deleted.

When an incident report is sent, it is immediately translated to XML with a simple function that retrieves that record’s information and converts it to an XML string based on the schema. It is then sent. When “Submit” is selected, the program sends the XML forms to the data transmission system, and blocks until an acknowledgement is received. If a successful acknowledgement is received for a form, the system deletes the data from the PDA. If an error is received, the system informs the user with instructions on how to resolve the error. See Figure 11: High Level Client Side Diagram.

3.2.2.2 Creating the Application

The client-side application was split up into two components, a data-entry system, and a data formatting system. The data-entry system allows the user to enter an incident report, all at once or piecemeal, and to use their PDA for any other task. This necessitated a state saving system, which simply, upon exit, saves the data already input, and allows it to be retrieved and completed upon reentry into the system. The data the user enters is able to be stored and edited as much as the user

desires, and will not be sent until the user deems it appropriate. As soon as the user instructs the application to submit the data, the client application invokes the transmission functions.

The data formatting system formats the inputted data into appropriate files to be transmitted to the database. Using a predefined XML schema, the data is transformed into an XML document and then passed on to the data transmission system. At this point, the system no longer handles the data nor is the user able to edit or change the data unless they do so through to the back end system.

3.2.2.3 Storing Data in the Client Application

The data entered into the application by the technician is, at first, stored persistently on the device itself. The Palm OS provides a Data Manager, which allows for the persistent storage of data between executions of an application. Due to the importance of the integrity of the data in the persistent storage areas, the Palm OS protects the memory in those areas by requiring any accesses to them to use Data Manager functions.

The Palm OS uses a database metaphor to manage access to persistent data. A Palm database contains header information and maintains links to records in the database. Unlike traditional relational databases, the Palm database format does not specify any particular method for accessing components of a record. A record is simply a reserved area of memory accessible using an index and a database pointer – the API user can define a record in any manner.

This project defines five databases, one to hold each of the following types of information: incidents, medications, allergies, vitals measurements, and responders. The incident database is responsible for storing information pertaining to each incident report submitted, including information relating to the incident's transmission status within the system. An incident record is defined as a C struct with a number of fields, which include a unique (within the Palm) incident number, data entered by users, transmission status, and submitter security information. An allergy record contains the number of the incident with which is associated and the name of the allergy as a string. A medication record similarly contains the incident number and medication string. The responder

database is populated via communication with the server, and stores information about the responders that can be associated with an incident.

3.2.2.4 Formatting Data for Transmission

When the data is ready to be formatted for transmission to the server, an XML string is generated from the corresponding incident record in the “incident” database on the device and sent to the server. Technicians can only send one incident report at a time, so the report sent is that which the technician selects from the main menu screen in the application.

3.2.2.5 User Interface Design

The interface decisions made for the Emergency Incident Reporting were designed to provide a fast, straightforward application that follows the Palm Operating System user interface guidelines (PALM03f). The design decisions documented in the following sections represent the best attempt to provide the Emergency Medical Technicians with an ideal application.

3.2.2.5.1 Main Form Layout

When users launch the application they are presented with the main form. This form is essentially used for the managing and creating of incident reports, but allows for the user to access and edit EIR-specific preferences as well.

The large box on the screen known as the incident list, displays currently open and un-submitted incidents. When an incident is added, the form displays it in the list, displaying incident number, the submitters ID number, and the last name of the patient for each incident (See Figure 12: Main Menu Form.) The user may click on a listed incident to open and edit it, when it is saved the display will update with the appropriate data. This method of data access and management provides a quick way to

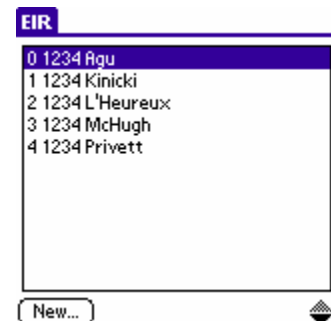


Figure 12: Main Menu Form

view and edit existing reports, without bogging the technician down with any unnecessary data.

To create a new incident, the user must select the “New...” button located on the bottom left of the screen. This is a standard button and location for Palm applications, and will allow the technician to create a new incident with a single tap. Creating a new incident will immediately send the user to the first data input screen, allowing the user to record data right away.

Preference commands are located in the drop down menus, which can be accessed by tapping on the title of the form, or the drop down button located in the graffiti area. In the “File” menu, named such since most application use this for important commands, users may set the server information or update the list of responders. These functions are rarely used, but still important, which lead to the decision to locate them in an out of the way place.

3.2.2.5.2 *Input Form Layout*

The group of forms used to input and edit data, known herein as “input forms”, share many components, these will be discussed in this section. The standardization was necessary for quick visual recognition of components, providing for a faster data entry. See Figure 13: An Input Form for an example.

Each input form contains a navigation bar, located at the top of the form, next to the form title. The bar consists of a left and right navigation arrow and a navigation drop down list. Using the arrows the user can progress or return to the next or previous input form, using the drop down navigation list the user may quickly jump to a desired form. Two navigation methods were provided to suit the use profile of a technician: the application needed to be designed to accommodate step by step data entry, as well as to allow the user to quickly access and edit a desired form.

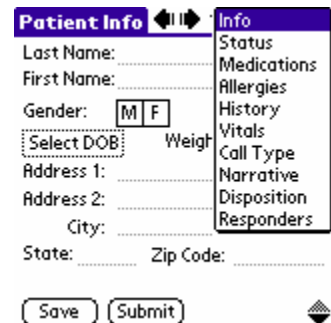


Figure 13: An Input Form

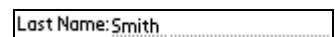
Running along the bottom of each form are form-specific commands. The “Save” button is used to save the edits made to a form, while the “submit” button saves and actually sends the data to the server. Upon successful submission of data, the program returns to the main form and deletes the submitted incident. To the right of the save and submit buttons is the shift indicator. This is a standard Palm component that indicates the current status of the Graffiti system, such as caps lock or punctuation mode. Immediately to the right of this, on appropriate forms, are the vertical scroll buttons. These buttons are used to scroll down each form and access non visible components. The placement of these buttons follows standard Palm UI guidelines.

The drop down menu for these components allows for several less frequently used commands to be executed. Under the File heading the user may delete the incident that they are currently editing, while Edit contains the standard edit commands, such as copy, cut, and paste. These functions are located out of the way not only to simplify the interface, but to prevent accidental activation of the delete command.

The body of each of the standard input forms contains four standard UI components; forms containing the non-standard table feature are covered in the next section. The four components are: text fields, selectors, push buttons, and check boxes. Each play a unique part in the system, they are described in the following sections.

3.2.2.5.2.1 Text Fields

Text fields are used to enter text from the graffiti or built in keyboard.



They appear as a dotted underline on the form, and can be edited by

Figure 14: A Text Field

tapping with the stylus on the field. Text fields are set to hold the maximum number of characters for each field, so the user may never overflow the field. These fields are used only where it is necessary that the user enter freeform text, other data entry methods are preferred since they minimize user keystrokes. See Figure 14: A Text Field.

3.2.2.5.2.2 Selectors

Selectors are used to activate features, most commonly the date or time selection dialogs. Selectors appear as dotted boxes surrounding text.

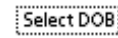


Figure 15: A Selector

The user activates them with a tap, which pulls up an appropriate form, and when the data is entered on the field the text in the selector is set to the entered data. These components are most often used to access standard palm components, but may be programmed any way the developer sees fit. In the EIR program, however, they are only used to access date and time selection forms. See Figure 15: A Selector

3.2.2.5.2.3 Push Buttons

When a set number of choices exist for a field, and the user must only select one, the push button is the ideal choice. A text field surrounded

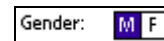


Figure 16: Push Buttons

by a solid box, when this box is tapped it turns solid black with white text, indicating selection. Push buttons may be set so that only one choice in a developer-defined group may be selected at a time, selection of a new button un-highlights the previous selection. These push buttons are very convenient for the user, as they present a quick method of entering data, as well as a large target to tap on. While push buttons can be used to select multiple values, to eliminate user confusion, they are only used as “radio”-style buttons. See Figure 16: Push Buttons.

3.2.2.5.2.4 Check Boxes

Check boxes provide a method to select multiple pre-defined choices, as opposed to push buttons which this program utilizes only as radio-

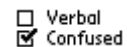


Figure 17: Checkboxes

buttons. Check boxes are represented as a small box next to a label; clicking on the box or label inserts a check into the box indicating selection. This provides a simple, familiar, and quick data method entry. See Figure 17: Checkboxes.

3.2.2.5.3 Database Control Form Layout

The Emergency Incident Reporting program provides the ability to enter an unlimited number of certain data types, including medications, allergies, and vital sign measurements. A special case of this occurs with the responder list, which is also described here.

Each form of this type expands on the functionality of a standard input form. To the right of the Submit button is located the “New” button. When tapped, this button pulls up an appropriate dialog in which the user enters information pertaining to one instance of the data type. When the user taps OK, the dialog is closed and the data is now displayed on the screen. The user may tap the list to edit entered data, and while editing use the drop down menu to delete the instance. The user may enter as many pieces of data as needed, using the vertical scroll bars to scroll through the list.

The responder form is a special case of this form type. Five tables are displayed; tapping on one brings up a list of available users. The user must tap on a responder, which is selected and displayed in the appropriate field back on the responder form. This allows the user to choose five total responders, including themselves.

3.3 Data Transmission

The second major phase of this project deals with the transmission of the incident report from the handheld device to a database located on a server, and the retrieval of a list of responders. This list of responders defines those who can both use the application to send incident reports and be selected

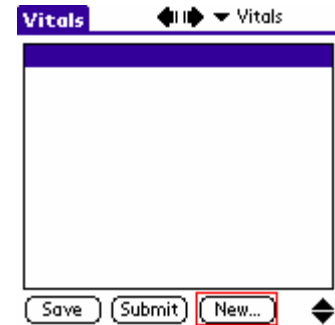


Figure 18: Database Control Form

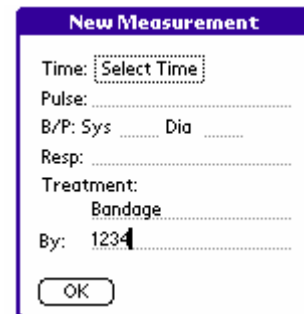


Figure 19: Database Entry Form

as responders to an incident. Both of these sets of data are transmitted over Worcester Polytechnic Institute's 802.11b wireless network.

Direct Serial Access, one of the methods for synchronizing data between a PC and a handheld device, requires a conduit, which is invoked by the HotSync Manager on the PC. The conduit is written and run by the HotSync Manager to enable synchronization between the two machines. Direct Serial Access will not be utilized in this project. In the case of this project, no data is being "synchronized." Data is merely being sent from one end to the other through TCP/IP protocols.

3.3.1 Wireless Transmission

The default method for transferring the incident report and responder table within the system is via a connection to WPI's wireless network. The Net Library provides networking services for the Palm OS and is utilized to establish connections and transfer data over the wireless network. See Figure 20: Connection Process

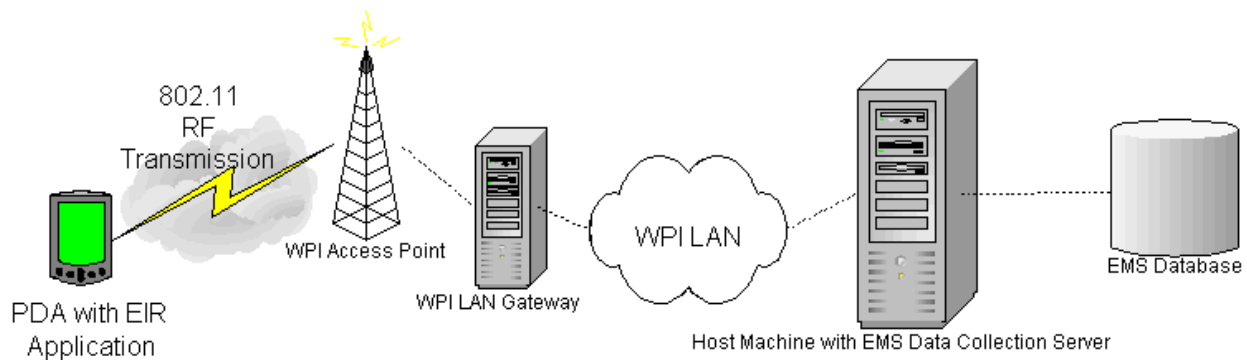


Figure 20: Connection Process

3.3.1.1 Sending the Incident Report

When the user wishes to send an incident report to the server, they select the "Submit" button on that report's form from within the application. A responder can only send one incident report at a time. Before the connection is made, the user is prompted for their password. As the connection process takes place, the user will see a window appear informing them that the NX60 is establishing

a connection to the wireless network. The user will then see a dialog box informing them that the incident report is being sent. When the transfer is complete, another window will appear informing them of the status of the transmission.

Several actions are taking place behind the scenes of the submit button. The client application calls functions that prepare and send the data. Essentially, these functions are a part of the client application in terms of implementation, but in terms of design and development, it is much easier to view these functions as a separate component.

First, a function that generates an XML string based on the information found in the “incident” database for the specific record being sent is called. When that string is generated, it is passed to `sendData()` which sends it to the server. This “send” function uses NetLib to find the server, open a connection between the client and server, and send the data to the server using TCP.

Status Code	Meaning
0	Success
1	I/O error
2	JDBC connection error
3	XML parser error
4	SQL error
5	Unknown submitter ID
6	Bad password

Table 3.1: Server status codes returned to client

Once a connection has been established, the application immediately sends as much data (usually all of it) as it can to the server. In the case that it doesn’t get all the data through on the first try, the function will loop through until all the data is sent. Upon retrieval of the XML string, the server will send an acknowledgement to the client detailing what happened on its end after it received the XML string. The client application

waits for an acknowledgement from the server. There are seven possible acknowledgements that the server could send, and a numeric value is associated with each. See Table 3.1: Server status codes returned to client for details about each status code.

If the client receives a successful acknowledgement from the server, the record that was sent is

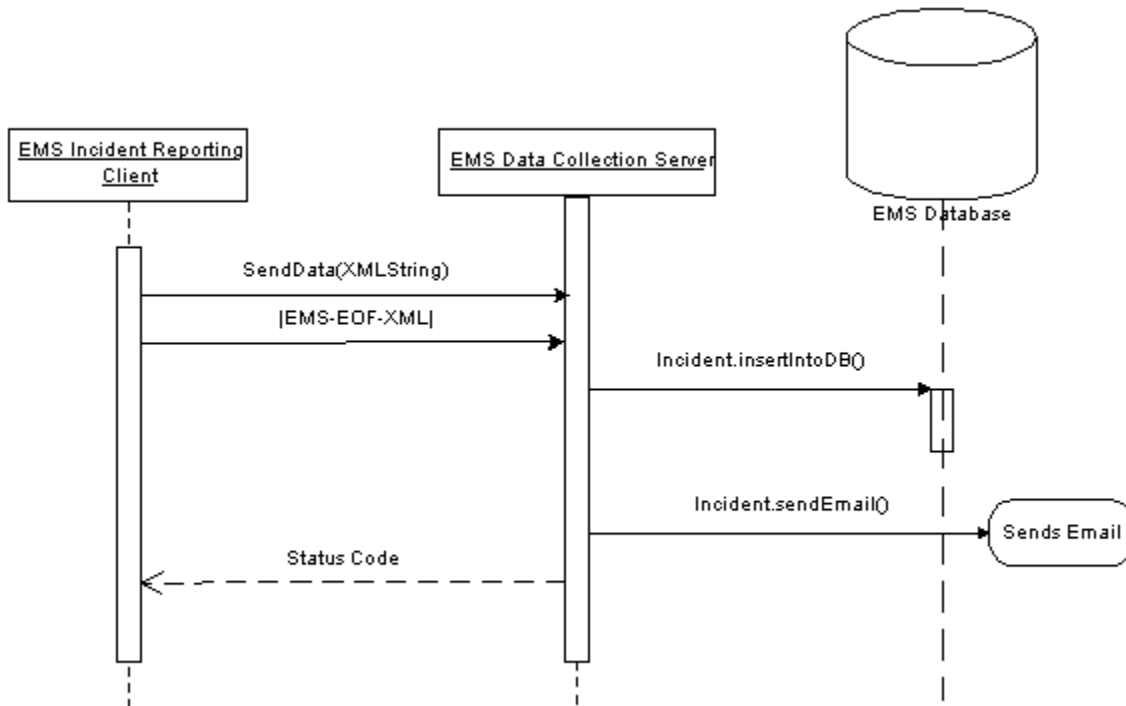


Figure 21: Sending an Incident Report

deleted from the “incident” database on the handheld. If it receives any other acknowledgement, it sends the user back to the form from which they were trying to send data. At this point, the user has the option of correcting any mistakes they could have made while completing the form, or they could try sending again. See Figure 21: Sending an Incident Report.

3.3.1.2 Receiving Responder Table

When the user wishes to update the list of responders on the handheld device, the user selects the “Update Responders” option from the main menu at the top-left corner of the initial screen of the application. As in the incident record transmission, while the connection is taking place, the user will see a window appear informing them that the NX60 is establishing a connection to the wireless network. When the transfer is complete, another window will appear informing them that the transfer has completed and the list of responders has been updated.

Prior to returning the current list of responders, the client application closes and deletes the existing “responder” database on the handheld device. There is no need to retain any previous responder list

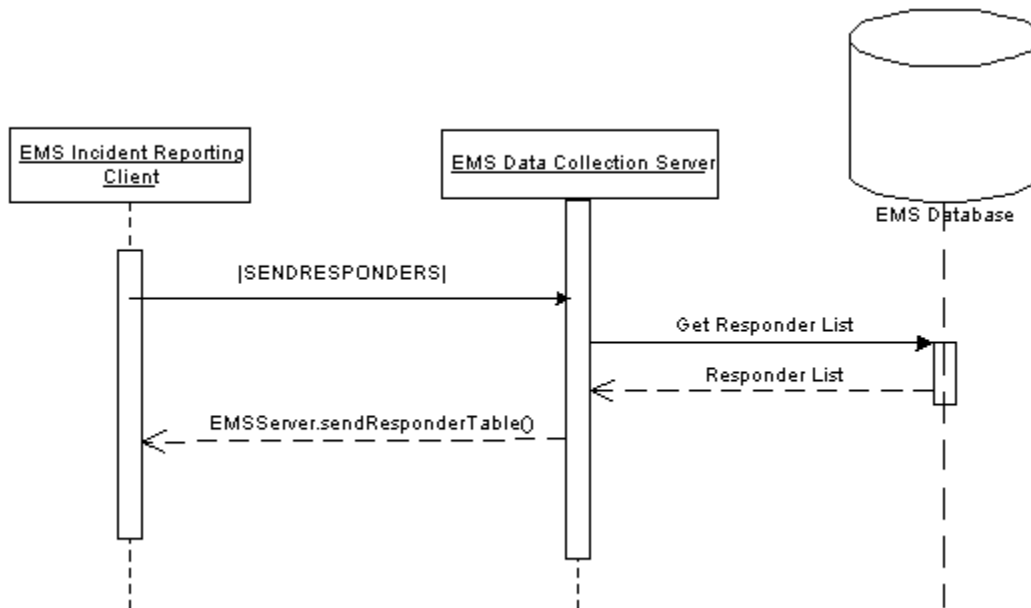


Figure 22: Receiving the Responder List

information since a new list is going to be inserted in place of the old one. The client software then re-opens the “responder” database (now empty) so that it can receive the incoming responder table information. The client application calls a function that sends a request string “|SENDRESPONDERS|” to the server. The server understands this string and realizes that it is in fact a request for the responder table, rather than an incoming incident report. When the server receives this message, it sends the responder table.

The responder table is a delimited list of responder information, the responder’s id, first name, middle name, last name, badge number, and role. Since the NetLib send and receive calls require the developer to specify the number of bits along with the transmission/reception, the incoming responder table had to be read in bit-by-bit, since it would be different each time and names of responders are always changing. Each delimiter was accounted for in the reception of the data, and when each important piece of the responder record was encountered, the function enters a loop until

the next delimiter is encountered. When each responder is retrieved, it is stored in the “responder” database on the handheld device. See Figure 22: Receiving the Responder List.

3.3.1.3 Encryption

Two layers of encryption are available to ensure the data is sent safely to the database. WEP encryption is needed to secure the transmission as it travels across the wireless medium. SSL encryption is needed in two cases; either the data reaches the wireless access point, or the HotSync is used to upload data to the server via a serial connection.

3.3.1.3.1 WEP Encryption

To ensure the maximum level of security, the highest level of WEP that is common between all devices in the wireless system was utilized.

To maximize your security, you should always utilize the highest level of WEP that your hardware supports. Sometimes, if you use hardware from several different vendors, you may find that they support varying levels of WEP. In these cases, you should use the highest level common to both devices. Although generally WLAN products from different vendors communicate with each other just fine, enabling WEP is often a way to expose interoperability problems. If security is your paramount concern, consider getting all of your hardware from a single vendor. (MORA02)

Enabling WEP on WLAN equipment is not difficult. Any WEP-enabled router, access point, or NIC has a WEP configuration section that allows the developer to specify the type of key they want to use as well as the key itself. Most devices enable the developer to specify a key using either ASCII (alphanumeric characters) or hex numerals (0-9 and A-F) (MORA02). Each device in the wireless system must have identical settings in terms of the key itself, and the key length. Each device must have WEP enabled; otherwise, they will not be able to communicate with each other.

For the NX60, WEP encryption is enabled when a user selects which type of connection they wish to make to their wireless LAN. From the main screen, the user can open the “Preferences” menu. They can furthermore select “Network – Preferences” in which they can select a type of connection, give that particular connection an ID, and choose which level of WEP encryption to enable, as well as set

the WEP key. In this project the highest level of WEP encryption that the NX60 would support, 128-bit, was used.

3.3.1.3.2 *SSL Encryption*

SSL encryption acts as a second layer of encryption to protect the data being sent from the wireless access point to the database server and from the HotSync to the database server. Because SSL is built into all major browsers and Web servers, simply installing a digital certificate turns on their SSL capabilities.

SSL encryption was not implemented in this project because was not applicable for two reasons. One of the major requirements of this project is to have incident report information sent to an appropriate EMS contact via email once it has been stored in the EMS database. Since WPI does not have an SSL enabled SMTP email server, there is no point in enabling SSL because once the email is in transit, it is vulnerable to any type of interception.

3.4 Server-Side Processing

The project required server-side support to enable the common storage of EMS run sheet data. This support consists of a server application and relational database implementation. See Figure 23: Server Overview.

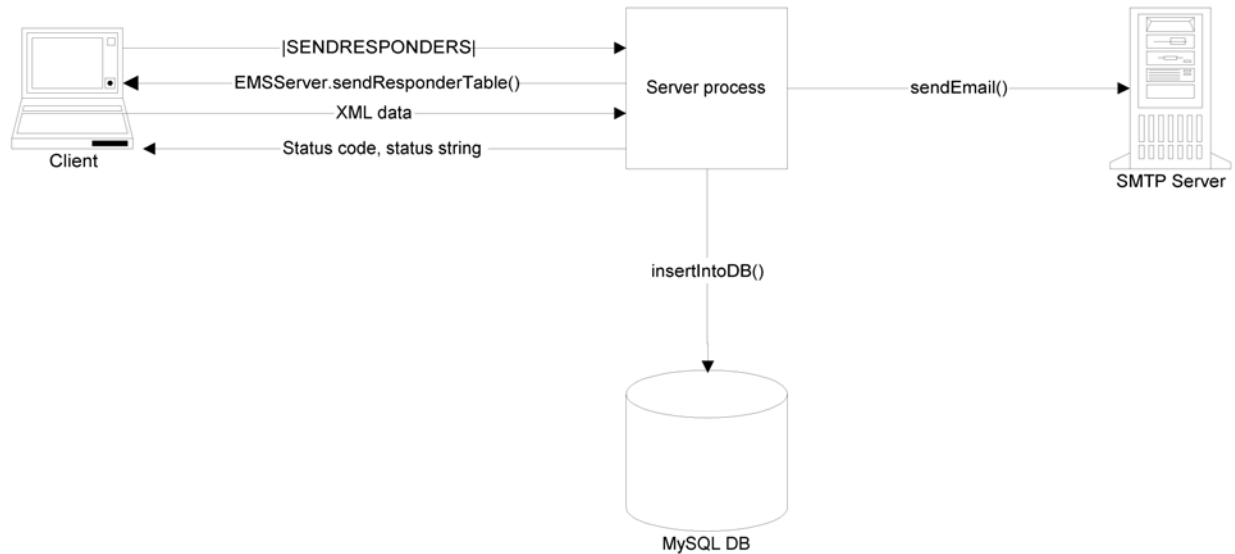


Figure 23: Server Overview

3.4.1 Data Processing Server

The server application is responsible for receiving the connection of the client application. This connection serves to receive the run sheet data from the client application. Once the data has been received, the server application is responsible for communicating with the database management system, and instructing the DBMS to insert the run sheet data from the handheld.

The server application was written in Java. This allows complete portability of the application after it has been written. Java also has a database connectivity API called “Java Database Connectivity” (JDBC). JDBC allows for a common database communication programming interface. This will allow the DBMS used to be changed without significant modification to the application. Additionally, the use of Java allows for a more rapid development of the server application than a language such as

C++ might allow. WPI's EMS department also requested the ability to have run sheet data emailed to certain contacts if necessary. This is a function that is also provided by the server application, by relaying email to an SMTP server for dissemination.

The server side of the application is responsible for obtaining data sent from the client and storing it in the database. Additionally, the server is responsible for providing status to the client application so that the client can act appropriately based on the status sent. The server application listens for connections from the client application. Once a connection has been requested, a secure socket is created. Through the secure socket, the client transfers an XML file to the server. After the XML file has been transferred, the socket remains open so that the server may provide final status to the client.

While the socket is open, the server processes the received XML data. It stores the XML file either as a Java object. The server application then parses the XML file to convert it to a Java object, an IncidentReport, a special class defined in this project. Once the data from the XML file has been converted to Java objects, the server formulates SQL statements that insert the run sheet data into the EMS database. After having formulated the SQL, the server opens a JDBC connection to the EMS MySQL database using the MySQL driver. It then sends the queries that update the database.

The server must provide confirmation to the client that the data has successfully been added to the database, the server process must verify that the data exists in the database. If the database insert was successful, the server sends a success acknowledgement back to the client and the connection is closed. If there was an error during the insert or receipt of data, an error code and description is also sent back to the client so that the client can respond appropriately.

3.4.1.1 Operating System Considerations

The use of Java, as mentioned, allows for operating system portability once the application has been written. The WPI EMS department currently has a Windows PC that it has allocated for use as a

server. The department granted us permission to load any OS on the PC, but the use of Windows will allow the PC to be used for other purposes than the server and database.

3.4.1.2 Data Translation

The main responsibility of the server application is to translate the data obtained from the client connection to data that is able to be inserted into the database. Because the XML data has a static schema, individual pieces of data were extracted and converted into Java objects more easily. Additionally, Sun is developing a technology (currently in beta) that allows automatic conversion of XML data to Java objects. This technology was utilized in the XML-to-Java conversions. If the use of the Sun technology had not been feasible, the static schema would have allowed us to parse the XML files and manually convert each piece of data into a variable or other object in Java.

3.4.2 Database Implementation

Once the data from the handheld is translated and converted to Java objects, the data needs to be stored in a database. The use of a database management system permits the ability to access the data itself, as well as to effectively store the information for later retrieval.

3.4.2.1 DBMS Selection

MySQL was the best choice as a DBMS for this project. Its primary advantages include attribute-level access control, a native JDBC connector, and an open-source license. Additionally, it is available for multiple platforms including Linux and Windows, allowing the system to be more platform-independent.

Because MySQL has a JDBC connector, it was possible to write for a single interface for any JDBC-supporting DBMS. After each piece of data from the run sheets was stored as Java objects or variables, connections to the database could be made. Using these connections, the data was inserted in the database by forming SQL query strings.

3.4.2.2 Data Design

The design of the database was reached through the standard normalization process once the necessary data had been finalized from the run sheets and reporting requirements of the EMS department. Entity-Relationship modeling was used in this process. To produce an Entity-Relationship (E-R) diagram of the data, each “entity” that exists in the data was shown on the diagram. For instance, an incident, a patient, and an officer could each be considered an entity. After the entities had been finalized, the relationships between them were formulated and represented on the E-R diagram.

After the E-R diagram had been finalized, each entity became a relation, or table, in the database. For instance, there is a relation for incidents. If there was a relationship between incidents and patients, each incident record would have referred to a record in the patient relation using a foreign key.

Because MySQL supports attribute-level permissions, a completely normalized data model without specific security considerations could be used. If attribute-level permissions were not available, there would have needed to have been a change in the design to completely isolate patient data or keys from the incident data.

3.4.2.3 Reporting

A required feature of the system was the ability to perform reporting on the data in the database. Reports on type-of-incident by time period and technician, for instance, were requested by WPI EMS. Microsoft Access provides the ability to “link” to tables in other DBMSes. Using Access; reports were easily produced using Access’s reporting features. See appendix 9.1 for examples of the current and prototype incident report forms.

3.5 Design Conclusion

Based on the design decisions set forth in the methodology, a successful implementation of the Emergency Incident Reporting System application was created that both suited the needs of the Emergency Medical Services as well as improved upon the existing process in use at the EMS office. The planning and design performed in this section allowed the application to be completed in an efficient manner, to user specifications, and to be submitted on time. The following section summarizes the methods used to implement the application set forth in this section.

4 Implementation

4.1 Client Application

4.1.1 Application Organization

The client side application is divided into two distinct portions: data entry and data storage. The user interface is designed and implemented as part of the data entry program, as well as the system for parsing and displayed entered information. The data storage section is simply in charge of handling all requests to the four databases used in this project.

4.1.2 Application Operation Overview

When the EIR application is launched, the user is faced with the main form. From this form they may perform four basic operations; they may create a new incident report, edit an existing incident report, update the responders list, or set the current server information. Figure 11: High Level Client Side Diagram.

Creating a new application creates a new record in the database, and then launches the input section of the program. Editing an existing record performs a similar action, opening the existing record into the input forms, allowing the user to see and edit the data they previously entered.

Updating the responders list calls the code to retrieve the list of valid responders from the server. Editing the server information loads a dialog window that allows the user to enter a server and port number to call on all network calls.

Once the user is in the input forms, they are able to browse sequentially, or navigate using the drop down bar. The technicians are free to fill in the data as they see fit, entering as much or as little at a time as they would like. Selecting the save option from any form saves all entered data and exits to the main form, the submit button saves the current data, transmits it to the server, and, pending a successful transfer, deletes the record from the database, and returns the user to the main form.

4.1.3 Event Driven Application Execution

When the Emergency Incident Reporting application is launched on the Palm, it simply loads the initial form and starts an event loop. This loop runs indefinitely, waiting for the operating system to report an event. When this event is received, the loop calls the event handling functions to attempt to handle. Unhandled events are just dropped.

An event is the direct result of any user tap on the screen. The stylus touching the screen triggers an event, any dragging movement creates events, and removing the stylus creates events. When the trigger is sensed to have touched a control, this triggers a special event dependant on the control selected. For example, if the user taps on a button, an event is generated, this event contains a wealth of information on the action, most importantly the name of the control tapped. The event handlers have the job of recognizing it as a button press, and then telling the program how to handle a tap on the particular control. Once the event handler is done performing any tasks related to the button, it returns to its loop. All events on the EIR application pass through this process, and all events are triggered in this manner.

4.1.3.1 Palm Main

Function	UInt32 PilotMain (UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
Purpose	This is the main entry point for the application.
Parameters	UInt16 cmd - word value specifying the launch code. Mem Ptr cmdPB - pointer to a structure that is associated with the launch code. UInt16 launchFlags - word value providing extra information about the launch.
Result	Result of launch
Called By	Palm OS upon launch of program.

PilotMain is the standard that must be present for a program to launch. Upon launch, the PalmOS calls the function and passes the launch parameters.

4.1.3.2 EIR Palm Main

Function	static UInt32 EIRPalmMain (UInt16 cmd, UInt16 launchFlags)
Purpose	This launches the event handler system
Parameters	UInt16 cmd - word value specifying the launch code. UInt16 launchFlags - word value providing extra information about the launch.
Result	Result of launch
Called By	Palm OS upon launch of program.

This is a standard function that actually launches the event handler system, which in turn controls the whole program.

4.1.3.3 AppStart

Function	static Err AppStart (void)
Purpose	Get the current application's preferences, and initialize variables.
Parameters	None
Result	0 if successful, error code if fails
Called By	EIR Palm Main

AppStart is called at the launch of the program to retrieve user preferences. This function also initializes certain variables needed during the application run.

4.1.3.4 AppStop

Function	static void AppStop (void)
Purpose	Save the current state of the application as well as close open databases.
Parameters	None
Result	0 if successful, error code if fails
Called By	EIR Palm Main

AppStop is called at the conclusion of each application run. This function saves preferences as well as closes any databases that have been opened during execution.

4.1.3.5 AppEventLoop

Function	static void AppEventLoop (void)
Purpose	This routine is the event loop for the application.
Parameters	None
Result	None
Called By	EIR Palm Main

The **AppEventLoop** waits for an event to occur, and then handles it by calling each successive event handler until the event is handled. This function is vital to the program, as it coordinates everything that happens during execution.

4.1.3.6 AppHandleEvent

Function	static Boolean AppHandleEvent (EventPtr eventP)
Purpose	This routine loads form resources and sets the event handler for the form loaded.
Parameters	eventP - a pointer to an EventType structure
Result	True if handled
Called By	AppEventLoop

If **AppEventLoop** receives an event without an event handler, it calls **AppHandleEvent** which sets up the event handler. If the form is defined, the program sets its handler and returns true. If the form does not find the form, it throws a fatal error and returns false. Every form in the program must be included in this function to be displayed properly and to have its events handled.

4.1.3.7 Event Handlers

Function	static Boolean MainFormHandleEvent (EventPtr eventP)
Purpose	This function handles events for any function whose event handler is set to MainFormHandleEvent
Parameters	eventP - a pointer to an EventType structure
Result	True if handled
Called By	AppEventLoop

Function	static Boolean InputFormHandleEvent (EventPtr eventP)
Purpose	This function handles events for any function whose event handler is set to InputFormHandleEvent. This function is the main event handler for the application.
Parameters	eventP - a pointer to an EventType structure
Result	True if handled
Called By	AppEventLoop

These functions attempt to handle events generated by the tapping of the screen. Events are passed into the function, the function then searches for a matching event definition to handle. If the event can be handled here the function returns true, if not it returns false. These functions, are called by **AppEventLoop** whenever it receives an event.

4.1.3.8 Menu Handlers

Function	static Boolean MainFormDoCommand (UInt16 command)
Purpose	This routine performs the menu command specified.
Parameters	command - menu item id
Result	True if handled
Called By	AppEventLoop

Function	static Boolean InputFormDoCommand (UInt16 command)
Purpose	This routine performs the menu command specified.
Parameters	command - menu item id
Result	True if handled
Called By	AppEventLoop

These functions are in charge of handling events generated by the tapping of a drop-down menu command. If the command was handled, the function returns true.

4.1.3.9 List Initializers

Function	void InitList (FormType * frmP)
Purpose	This function initializes the main list form.
Parameters	frmP - pointer to the current form.
Result	None
Called By	MainFormHandleEvent

Function	void InitMedList (FormType * frmP)
Purpose	This function initializes the med list form.
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

Function	void InitAllergyList (FormType * frmP)
Purpose	This function initializes the allergy list form.
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

Function	void InitResponderLists (FormType * frmP)
Purpose	This function initializes the responder lists
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

Function	void InitResponderList (FormType * frmP)
Purpose	This function initializes the responder selection list
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

Function	void InitMeasurementList (FormType * frmP)
Purpose	This function initializes the measurement list form.
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

These functions handle the initialization and filling of the lists on their corresponding forms. Each function searches through its database to find the data to display, display all records that can be shown at the moment. This function also handles the scrolling of the database lists. When the event handler receives that one of these forms has been opened, it calls the appropriate initialization function.

4.1.3.10 Parsing Forms

Function	Boolean InputFormParseData (FormPtr frmP)
Purpose	This routine parses data from the passed form. If the form is unrecognized, the routine throws a fatal exception
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

When a form is exited, either through the navigation buttons or through the save button, the form is parsed using **InputFormParseData**. This function simply calls a function based on the current form, these functions simply take the text from each field and store it in the database.

4.1.3.11 Loading Forms

Function	Boolean InputFormLoadData (FormPtr frmP)
Purpose	This routine parses data from the passed form. If the form is unrecognized, the routine throws a fatal exception
Parameters	frmP - pointer to the current form.
Result	None
Called By	InputFormHandleEvent

Similar to **InputFormParseData**, this function simply calls the appropriate function to load the current information with the stored data, if any exists.

4.1.4 Writing to the Palm Databases

The Palm OS requires that all accesses of persistent memory be performed through the Data Manager functions, to protect the integrity of the data. As such, there is some overhead that is required when manipulating records in the databases. A number of functions were implemented to handle this overhead so that the database implementation details can be ignored in the other parts of the application.

The Palm Data Manager provides a variety of functions, including functions to retrieve a handle to a record based on its database reference and index, to delete a record using the same information, and to create new records in the database. All writing to a database must be performed through the

DmWrite call, which ensures that a record has been locked for editing and that it is in a valid location. These functions form the basis of the project’s database helper functions.

4.1.4.1 Creating the databases

When the application starts for the first time, it must create the four databases that it uses. To do this, a call is made to **EmsOpenDatabase**. If the database does not yet exist, the database is created; otherwise, the database is simply opened and the DmOpenRef passed to the function is set to point to the opened database.

The DmOpenRef type is an important component of using the Data Manager. Once a DmOpenRef has been initialized using **EmsOpenDatabase**, it stays pointed to its original database and can be used perpetually to access that database. Nearly all database functions take a DmOpenRef as an argument.

Function	Err EmsOpenDatabase (DmOpenRef *dbP, char* emsDBName, UInt16 mode)
Purpose	Opens a database with the given name. If the database does not exist, the database is created and opened.
Parameters	dbP – database pointer to use to access the database after it has been opened emsDBName – name to apply to the new database mode – mode to open the database in (read only or read-write)
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from the AppStart() function in eir.c

Function	Err EmsCreateDatabase (char* emsDBName, UInt16 mode)
Purpose	Creates a database with the given name
Parameters	emsDBName – name to apply to the new database mode – mode to open the database in (read only or read-write)
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from the EmsOpenDatabase() function in eirDB.c

In addition to initializing the databases, the application must also set up a data storage area called the “application info block.” This area is used to store information pertaining to an entire application (as opposed to a single record) and must persist across executions of the application. In the case of

this application, the application info block stores the next unique ID to assign, the hostname of the server to connect to, as well as the port on which to connect to the server.

Function	Err EmsCreateAppInfoBlockForIncidentDB (DmOpenRef *db)
Purpose	Creates an application info block and attaches it to the given database
Parameters	db – pointer to the database to create info block for
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from the AppStart () function in eir.c

Access to information contained in the application info block is performed through the Data Manager functions, so functions were implemented to simplify these operations within the application. For instance, **EmsGetNextUniqueID** allows the application to easily find the next unique ID to assign.

Function	UInt16 EmsGetNextUniqueID (DmOpenRef *db)
Purpose	Obtains the next unique ID available for assignment from the application info block
Parameters	db – pointer to the database to create info block for
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from a number of functions in eir.c

4.1.4.2 Creating new records

When the creation of a new record has been requested by the application, **EmsNewIncidentRecord** is called. This function allocates new memory in the database using **DmWrite**. The client can then write to the new record using **EmsStoreIncidentRecord**. New and Store functions also exists for each of the other databases used by the application. Each of these functions makes use of the **DmWrite** call to copy data from the passed record structure into the data manager memory. Before writing, the memory pointer being written into is locked, and the record is marked busy. After the write, the record is released using **DmReleaseRecord**.

Function	Err EmsNewIncidentRecord (DmOpenRef *db, incidentRecord *record)
Purpose	Allocates a new record in the database and increments the next unique ID in the application info block
Parameters	db – pointer to database to create new record in record – pointer to record structure to copy into record memory
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from a number of functions in eir.c

Function	Err EmsStoreIncidentRecord (DmOpenRef *db, incidentRecord *record, UInt16 index)
Purpose	Stores the given record in the space denoted by the given index
Parameters	db – pointer to database to create new record in record – pointer to record structure to copy into record memory index – the index telling which record to store into
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from a number of functions in eir.c

4.1.4.3 Record sorting

Traversal of Palm databases depends on the assumption that deleted records have been moved to the end of the database. The Palm Data Manager provides a function, **DmQuickSort**, that, when given a callback function that compares two records, performs a quicksort on the database. During this operation, all deleted records are moved to the end of the database. This allows the safe traversal of the database record-by-record using the **DmNumRecordsInCategory** function, which returns the number of non-deleted records in the database. The callback function for the incidents database, **EmsIncidentCompareByUniqueID** compares records based on their unique ID. The functions for the medication and allergy databases compare the records based on the incident ID associated with each record.

Function	Int16 EmsIncidentCompareByUniqueID (void *r1, void *r2, Int16 unusedInt16, SortRecordInfoPtr unused1, SortRecordInfoPtr unused2, MemHandle appInfoH)
Purpose	Determines the sort order of two incident records
Parameters	r1 – pointer to the first record structure being compared r2 – pointer to the second record structure being compared
Result	1 if record 1's unique ID is greater than that of record 2, -1 if record 2's unique ID is greater than that of record 1
Called By	This method is called from EmsDeleteIncidentRecord() in eirDB.c

Function	Int16 EmsAllergyCompareByIncidentNum (void *r1, void *r2, Int16 unusedInt16, SortRecordInfoPtr unused1, SortRecordInfoPtr unused2, MemHandle appInfoH)
Purpose	Determines the sort order of two allergy records
Parameters	r1 – pointer to the first record structure being compared r2 – pointer to the second record structure being compared
Result	1 if record 1's incident number is greater than that of record 2, -1 if record 2's incident number is greater than that of record 1, 0 if the records' incident numbers are equal
Called By	This method is called from EmsDeleteAllergyRecord() in eirDB.c

4.1.4.4 Record deletion

Record deletion is a relatively simple process. When a record needs to be deleted, **EmsDeleteIncidentRecord** is called. This function calls the built-in **DmDeleteRecord** function, and calls **DmQuickSort** to sort the deleted record to the end of the database.

Function	Err EmsDeleteIncidentRecord (DmOpenRef *db, UInt16 index)
Purpose	Deletes the record in the given database at the specified index and sorts the database after the deletion.
Parameters	db – pointer to the database from which to delete the record index – the index of the record being deleted
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from a number of functions in eir.c

Because the application deals with a number of databases, some of which are linked, a function was required that deletes all of the records associated with a given incident. This is a more complex process than simple record deletion, as each database has to be searched for any records associated with the unique incident ID being deleted. This is done by iterating through the databases and obtaining pointers to each record in those databases. Within each record, the incident number is compared to the incident number being deleted, and if there is a match, **EmsDeleteMedicationRecord**, **EmsDeleteAllergyRecord** or **EmsDeleteMeasurementRecord** is called, depending upon which database is being searched.

Function	Err EmsDeleteRecordsByIncidentNum (DmOpenRef incidentDb, DmOpenRef measurementDb, DmOpenRef allergyDb, DmOpenRef medicationDb, UInt16 index)
Purpose	Deletes all records associated with the incident record denoted by the index
Parameters	incidentDb – pointer to the incident database measurementDb – pointer to the measurement database allergyDb – pointer to the allergy database medicationDb – pointer to the incident database index – the index of the incident record being deleted
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from the HandleStatusCode() in eir.c

4.1.4.5 Record retrieval

When an incident record needs to be retrieved, the application calls **EmsRetrieveIncidentRecord** and passes it a pointer to an existing record structure into which the data from the retrieved record should be copied. Similar functions exist for the allergy, medication, and measurement databases.

Function	Err EmsRetrieveIncidentRecord (DmOpenRef *db, incidentRecord *record, UInt16 index)
Purpose	Copies the data from the requested record into the incidentRecord structure passed
Parameters	db – pointer to the incident database record – pointer to an incidentRecord structure to copy the retrieved data into index – index of the record to retrieve
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from a number of functions in eir.c

4.2 Data Transmission

The transmission component of the system provides the functionality to send data from and receive data to the handheld device over WPI's wireless network. The client application consists of a file eirXMIT.c which contains all the functionality needed to send and receive data. All NetLib and System function are referenced via the Palm API Reference (PALM03g).

4.2.1 Wireless Transmission (NetLib)

The transmission component consists of two major functions that are a part of the client application, **SendData** and **GetResponderTable**.

Function	UInt16 SendData (Char *theXMLString)
Purpose	Sends an XML string (incident report) to the server and receive an acknowledgement.
Parameters	theXMLString – String containing the XML string corresponding to the incident report that is being sent
Result	Returns status of the transmission, 0-6, -1 otherwise
Called By	This method is called from the InputFormHandleEvent function in eir.c

Function	Err GetResponderTable ()
Purpose	Gets responder table and stores each record in the “responder” database.
Parameters	None
Result	0 if no error; otherwise: error code indicative of error that occurred
Called By	This method is called from the MainFormDoCommand function in eir.c

A collection of Net Library functions are used to give the desired functionality to these functions. Each of these Net Library functions is defined in NetMgr.h.

4.2.1.1 SendData

The **SendData** function sends an XML string containing incident report information. It is the function that is called when a user wishes to send an incident report. To send the string, the Net Library must be opened so the NetLib function calls can be utilized. With the Net Library open, **SendData** can verify a host name and service name and make a socket connection to the server. At this point, data can be sent or received. When finished with all network transactions, the socket, as well as the Net Library, is closed, and the status of the transmission is returned.

4.2.1.1.1 Find/Open Net Library

The Net Library must first be found using the **SysLibFind** system function before it can be opened.

Function	Err SysLibFind (const Char *nameP, UInt16 *refNumP)
Purpose	Return a reference number for a library that is already loaded, given its name.
Parameters	nameP – Pointer to the name of a loaded library. refNumP - Pointer to a variable for returning the library reference number (on failure, this variable is undefined)
Result	0 if no error; otherwise: sysErrLibNotFound (if the library is not yet loaded), or another error returned from the library's install entry point.
Called By	This method is called from the SendData() function in EIRXMIT.c

When the Net Library has been found, it is then opened by calling **NetLibOpen**. This call reads from the network configuration file on the handheld, making a physical connection to the wireless medium via one of WPI's wireless access points. The function then loads the network protocol stack.

Function	Err NetLibOpen (UInt16 libRefnum, UInt16 *netIFErrsP)
Purpose	Opens and initializes the Net Library.
Parameters	libRefnum – Reference number to the Net Library. netIFErrsP – First error encountered when bringing up network interfaces.
Result	0 - if no error netErrAlreadyOpen – returned if library is already open netErrOutOfMemory – not enough memory available to open library. netErrNoInterfaces – incorrect setup. netErrPrefNotFound – incorrect setup.
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.2 *Verifying Host Name*

Once the Net Library has been found and opened, the next step is to verify a host name. This is done by calling **NetLibGetHostByName**. A string variable containing the string location of the host is passed to this function. The host address is then stored in a variable which is passed into functions that connect the socket.

Function	NetHostInfoPtr NetLibGetHostByName (UInt16 libRefNum, const Char *nameP, NetHostInfoBufPtr bufP, Int32 timeout, Err *errP)
Purpose	Looks up a host IP address given a host name.
Parameters	libRefNum – Reference number to the Net Library. nameP – Name of host to look up. bufP – Pointer to a NetHostInfoBufType struct in which to store the results of the lookup. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is 0.
Result	Returns a pointer to the NetHostInfoType portion of bufP, which contains results of the lookup. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

After the host has been retrieved, the service port is set. This is done by using the **NetLibGetServByName** function. **SendData** calls this function using the “tcp” protocol type passed in as a parameter.

Function	NetServInfoPtr NetLibGetServByName (UInt16 libRefNum, const Char *servNameP, const Char *protoNameP, NetServInfoBufPtr bufP, Int32 timeout, Err *errP)
Purpose	Looks up the port number for a standard TCP/IP service, given the desired protocol.
Parameters	libRefNum – Reference number to the Net Library. servNameP – Name of the service to look up. protoNameP – Desired protocol to use, either “udp” or “tcp” bufP – Pointer to a NetServInfoBufType struct in which to store the results of the lookup. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is 0.
Result	Returns a pointer to the NetServInfoType portion of bufP, which contains results of the lookup. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.3 *Opening a Connection*

Now that the host machine and service port have been found, a socket can be opened. **NetLibSocketOpen** is called to achieve this task. A call to this function allocates resources that the socket will need.

Function	NetSocketRef NetLibSocketOpen (UInt16 libRefnum, NetSocketAddrEnum domain, NetSocketTypeEnum type, Int16 protocol, Int32 timeout, Err *errP)
Purpose	Opens a new socket.
Parameters	libRefnum – Reference number to the Net Library. domain – Desired type of connection. protocol – Protocol to use. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is 0.
Result	Returns the NetSocketRef of the opened socket or -1 if an error occurred. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.4 *Connecting Socket*

An open socket is not enough to send and receive data. The socket must be connected for any data to be sent and received. **NetLibSocketConnect** is called to connect the socket.

Function	Int16 NetLibSocketConnect (UInt16 libRefnum, NetSocketRef socket, NetSocketAddrType *sockAddrP, Int16 addrLen, Int32 timeout, Err *errP)
Purpose	Assign a destination address to a socket and initiate a three-way handshake.
Parameters	libRefnum – Reference number to the Net Library. socket – Descriptor for the open socket. sockAddrP – Pointer to address to connect to. addrLen – Length of address in sockAddrP. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is 0.
Result	Returns 0 upon success and -1 if an error occurred. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.5 *Sending Data*

At this point, **SendData** has all it needs to send the XML string to the server. The string is stored locally in a variable that is passed to the **NetLibSend** function, which sends the string to the server. Two functions to send data exist in the Net Library. This function was chosen to implement because data is being sent from a single buffer across the socket. The alternate function allows for data to be sent from multiple buffers. **NetLibSend** is given a specified number of bytes to send, which is the length of the XML string in bytes, and is placed within a while loop to ensure each byte of data is sent. After the XML string is sent, another string is sent (“|EMS-EOF-XML|”) to the server to signify the termination of the XML string, which lets the server know to process the XML string it just received. Both strings are sent via two separate instances of the **NetLibSend** function.

Function	Int16 NetLibSend (UInt16 libRefNum, NetSocketRef socket, void *bufP, UInt16 bufLen, UInt16 flags, void *toAddrP, UInt16 toLen, Int32 timeout, Err *errP)
Purpose	Sends data to a socket from a single buffer.
Parameters	libRefNum – Reference number to the Net Library. socket – Descriptor for the open socket. bufP – Pointer to data to write to. bufLen – Length of data to write. flags – One or more netIOFlagxxx flags. toAddrP – Address to send to (pointer to a NetSocketAddrType), or 0 toLen – Size of toAddrP buffer. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is -1.
Result	Returns number of bytes sent. Returns 0 if the socket has been shut down, and -1 if an error occurred. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.6 *Receiving Data*

Upon retrieval of the XML data, the server will process the string and send back a code indicating the status of the server as explained in section 3.3.1.1. **NetLibReceive** is used to get this status code from the server. This function needs only to read one byte of data as the number corresponding to the status of the server is the first piece of data sent back in acknowledgement.

Function	Int16 NetLibReceive (UInt16 libRefNum, NetSocketRef socket, void *bufP, UInt16 bufLen, UInt16 flags, void *fromAddrP, UInt16 *fromLenP, Int32 timeout, Err *errP)
Purpose	Receive data from a socket into a single buffer.
Parameters	libRefNum – Reference number to the Net Library. socket – Descriptor for the open socket. bufP – Pointer buffer to hold received data. bufLen – Length of bufP buffer. flags – One or more netIOFlagxxx flags. fromAddrP – Pointer to buffer to hold address of sender (NetSocketAddrType) fromLenP – On entry, size of fromAddrP. On exit, actual size of returned address in fromAddrP. toLen – Size of toAddrP buffer. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is -1.
Result	Returns number of bytes successfully received. Returns 0 if the socket has been shut down, and -1 if an error occurred. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

SendData takes the status and acts upon it accordingly. If status comes back as successful, the record corresponding to the incident record XML string is deleted as there is no need for it to remain in the handheld's database. If status codes 2 through 6 are received, the information was received correctly, but either it is invalid, or there was an error on the server's end in trying to parse the data.

In these cases, the record will remain the database on the handheld, which allows for the user to try sending it again. If status codes 5 or 6 are received, the server did not even try to put the XML data into the database. The record will remain in the database on the handheld, and the user will be sent to the “responder” form so that they can enter in the information needed to ensure a valid technician is using the device.

4.2.1.1.7 *Disconnecting the Socket*

When the socket is no longer needed, it is disconnected using the **NetLibSocketShutdown** function, which tells the network protocol stack that the application is finished with the network connection. **SendData** shuts down the socket in both directions.

Function	Int16 NetLibSocketShutdown (UInt16 libRefnum, NetSocketRef socket, Int16 direction, Int32 timeout, Err *errP)
Purpose	Receive data from a socket into a single buffer.
Parameters	libRefnum – Reference number to the Net Library. socket – Descriptor for the open socket. Direction – Direction to shut down. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is -1.
Result	Returns 0 upon success and -1 if an error occurred. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.8 *Closing the Socket*

With the socket shut down, it is no longer needed. At this point, **SendData** closes the socket with **NetLibSocketClose**. This closes the socket and releases all of the resources that the socket was using.

Function	Int16 NetLibSocketClose (UInt16 libRefnum, NetSocketRef socket, Int32 timeout, Err *errP)
Purpose	Closes a socket.
Parameters	libRefnum – Reference number to the Net Library. socket – Descriptor for the open socket. timeout – Maximum timeout in system ticks; a value of -1 will wait forever errP – Contains an error code if the return value is -1.
Result	Returns 0 upon success and -1 if an error occurred. errP contains the error if one exists
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.1.9 Closing the Net Library

The Net Library is no longer needed if there are no more network transactions taking place.

NetLibClose is called to unload and terminate the network protocol stack.

Function	Err NetLibClose (UInt16 libRefnum, UInt16 immediate)
Purpose	Closes the Net Library
Parameters	libRefnum – Reference number to the Net Library. immediate – If true, will shut down the Net Library immediately. If false, the library will shut down only if a close timer expires before another NetLibOpen is issued.
Result	Returns 0 upon success and error otherwise.
Called By	This method is called from the SendData() function in EIRXMIT.c

4.2.1.2 GetResponderTable

To get the responder table, the user instantiates a request to update the responder list through the main menu of the application. Because the existing “responder” database is being replaced each time a request is made to update the list, **GetResponderTable** first closes and deletes the existing “responder” database using **DmCloseDatabase** and **DmDeleteDatabase** (declared in DataMgr.h).

Function	Err DmCloseDatabase (DmOpenRef dbP)
Purpose	Closes a database on the handheld device.
Parameters	dbP – Database access pointer.
Result	Returns errNone if there is no error, or dmErrInvalidParam if an error occurs.
Called By	This method is called from the GetResponderTable() function in EIRXMIT.c

Function	Err DmDeleteDatabase (UInt16 cardNo, LocalID dbID)
Purpose	Deletes a database and all of its records.
Parameters	cardNo – Card number the database resides on. dbID – Database ID.
Result	Returns errNone if there is no error, an error code corresponding to the type of error.
Called By	This method is called from the GetResponderTable() function in EIRXMIT.c

GetResponderTable re-opens the “responder” database using **DmOpenDatabase**, which is discussed in section 4.1.4 above. The open database is now empty and ready to accept a new list of responders. **GetResponderTable** updates the list of responders by sending a request string to the server for the responder table (“|SENDRESPONDERS|”) using **NetLibSend** as mentioned in section 4.2.1.1.5, and then receives the incoming list of responders. Each time the function receives the

appropriate information for a single responder, a new record is created in the “responder” database on the handheld with that responder’s information.

The responder table is sent as a string of data from the server to the client application. Each field of the responder record is delimited by a tab (“\t”) character and each record is delimited by a new line (“\n”) character. With this being the case, and since responder name lengths aren’t always the same length, the incoming responder table string is read in 1 character (byte) at a time. This ensures that each character is accounted for, and there is no loss of data. **NetLibReceive** mentioned in section 4.2.1.1.6 is called to get the responder table string. It is stored locally in a variable that is then parsed to get each individual field of data. “While” loops are used as an index traverses the string which gathers and stores appropriate data in the responder table structure in the client application. Once the responder structure is complete for a single record, a new record is created in the “responder” database on the handheld with that record structure’s information using **EmsNewResponderRecord** defined in `eirDB.c`, which is explained in section 4.1.4.2 above.

4.3 Server Application

4.3.1 Managing settings

The server application requires several pieces of user-defined configuration information to run successfully. This information is necessary for performing the database connection, emailing incident reports, and ensuring connection availability. To manage the loading of this configuration information, there is a `Preferences` class that encapsulates all operations relating to configuration management.

The `Preferences` class is patterned on the Singleton object-oriented design pattern (GAMM95). As such, only one instance of it may be created and accessed during the execution of the server application. This ensures that configuration information is consistent. To implement this design pattern, the constructor is made private, and the class stores an instance of itself. The only way to create an instance of the class is to call the static method **`getInstance`**. The **`getInstance`** method

checks to see if an instance of the object has already been created. If not, it calls the constructor and stores the instance within the class. Subsequent calls to **getInstance** return the existing instance stored within the class, thus ensuring that only one instance of the class is ever created.

Upon startup, the EMSServer constructor calls the Preferences class' **loadAllPrefs** method. This method, in turn, calls individual load functions to load each piece of configuration information from a text file. For instance, to load the database hostname, **loadDBHostName**(String fileName) is called, where fileName is the name of the text file containing the property. Figure 24 contains information regarding the specific files and their functions.

File Name	Function
dbhostname.txt	Hostname where DBMS can be found
dbusername.txt	Username with which to access DBMS
dbpassword.txt	Password with which to access DBMS
dbport.txt	MySQL port
dbname.txt	Database name within DBMS
mailfromaddress.txt	Address from which emailed reports should appear to be sent
mailtoaddresses.txt	Addresses to which to send reports
serverport.txt	Port on which to run the server application
smtphostname.txt	Hostname of SMTP server to use for sending email reports
smtppauth.txt	SMTP username and password

Figure 24: Files loaded by the configuration manager

To access configuration settings, users of the Preferences class must use **get<PrefItem>** methods, which return the requested information.

Function	void loadAllPrefs ()
Purpose	Loads all preference settings from configuration text files
Parameters	None
Result	None
Called By	This method is called from EMSServer's constructor

Function	void loadDBHostName (String fileName)
Purpose	Loads the database hostname to be used to connect to the MySQL database
Parameters	fileName – name of the file containing the data
Result	None
Called By	This method is called from the loadAllPrefs () function

4.3.2 Handling connections

The `EMSServer` class is responsible for managing the continuing operation of the server. Upon creation by the server application's `main` method, the `EMSServer` class creates a `ServerSocket` object, which listens for incoming connections on port 1612.

When a connection request is received by the server, a new `Connection` object is created, which runs on a new thread. The `Connection` object then waits for input from the client. If the client sends the server the line “`|SENDRESPONDERS|`”, the `Connection` invokes the `EMSServer`'s `sendResponderTable` method, which is detailed in section 4.3.3 below. If any other input is received, the server assumes that the client is sending it an incident report via XML and begins storing the XML data received.

The “`|`” character was chosen to flag the beginning and end of server-client commands because this character is not reproducible using the Palm's built-in Graffiti input system. Additionally, because the data in each field appears as part of an XML tag, a user is not capable of sending the server only the command on a single line. If a user is able to copy a pipe character from a HotSync into the field, the presence of the XML tags and the escaping of newline characters prevents any user from directly issuing commands to the server.

When the server receives the line “`|EMS-EOF-XML|`”, it stops listening to client input and creates an XML `InputSource` object based on the XML string received from the client. This process is described in section 4.3.4 below.

4.3.3 Managing Responders

When requested by the client, the server is responsible for sending an up-to-date list of the responders that are authorized to submit incident reports and respond to incidents. To ensure the integrity of the incident reports received, each incident report must contain a responder number and the password corresponding to that responder number. It was decided that the best unique identifier

for responders was an incremental unique ID assigned by the database. Badge numbers are not suitable because they may change when a responder changes rank. Because this unique ID has no correlation to the real world, the system provides the ability for responders to select their name from a menu without having any direct contact with the unique ID that they were assigned.

To send the list of responders to the client, the **sendResponderTable** method first opens a JDBC connection to the MySQL database. It then performs an SQL query to determine the number of responders that exist in the database. This number is sent to the client to prepare the client to receive the correct number of responders.

The method then performs an SQL query to get the actual responder information, including unique ID, First Name, Last Name, Badge Number, and Role (office or responder). For each responder retrieved, the server sends the client the following:

```
RESPONDER_NUM\tRESPONDER_NAMEF\tRESPONDER_NAMEM\tRESPONDER_NAMEL\tRESPONDER_BA
DGENUM\tRESPONDER_ROLE\n
```

Thus, an example of a server transmission of two responders in response to a client's |SENDRESPONDERS| command may look like the following:

```
2\n1\tBrian\tRoss\tL'Heureux\t51\R\nMichael\tJames\tMcHugh\t52\tO\n
```

Function	public static void sendResponderTable (PrintWriter out)
Purpose	Sends the list of responders to the specified PrintWriter
Parameters	out – PrintWriter to send output to
Result	None
Called By	This method is called from the Connection's run() method when the client sends " SENDRESPONDERS "

4.3.4 Parsing XML Data

The server's XML parsing component is based on the Java-based "Simple API for XML" (SAX). This API is based on a Java interface that users of the API are responsible for implementing. This interface is equivalent to requiring that users implement handlers for each event the parser encounters while reading the XML. For instance, when the parser encounters the start of an XML

tag, it calls the **startElement** method that is implemented by the API user. This allows the user of the API to deal with the XML events as appropriate.

To store the information obtained from parsing the XML data received from the client, the server application contains an `Incident` object. The `Incident` object contains fields for data storage, and operations relating to individual incident reports.

When the **startElement** method is called, it determines the name of the tag found. If the tag name is one of those that requires special treatment (i.e. has attributes that need to be handled), that handling is done within the **startElement** function. In the server application, this tag is then set as the “current” tag. This designation is used for character processing.

After encountering an element tag, the parser will generally encounter character data. To handle this event, the SAX parser calls the **characters** method. Depending upon which tag is the current tag, the **characters** method concatenates each character encountered to the existing data stored in the `Incident` object under that variable.

There is some incident data that requires special handling, particularly those pieces of data that can have multiple instances per incident. One example of this is measurement data. For each incident, there may be multiple measurements that are taken. There is no logical limit to the number of measurements that can be taken for a given incident, so a `Measurement` class was created to store measurement data. These `Measurement` objects are related to the `Incident` objects through stacks. Each time a `<measurement>` tag is encountered by the parser, a new `Measurement` object is pushed onto the `Incident` object’s measurements stack. This `Measurement` object is subsequently populated by the parser as it encounters additional measurement data within the `<measurement>` instance.

4.3.5 Database Insertion

The logic for inserting incident data into the server's database is contained in the Incident object's **insertIntoDB** method. This method opens a Java Database Connectivity (JDBC) connection to the MySQL database specified by the configuration manager.

Once the connection has been made, the server first verifies that the submitter is authorized to submit an incident, based on the submitted user ID and password. To perform this check, a `SELECT COUNT(*)` query is performed to see if any entries in the `RESPONDER` table match the submitted user ID. If no match is found, an appropriate error is returned to the client, as described in section 4.3.7, and the connection is closed without the incident being stored. If a match is found, another query is performed to see if the submitted password matches that stored for the submitter in the `RESPONDER` table. If this check fails, the client is returned an error code and the connection is closed without the incident being submitted.

To begin the incident data insertion, an SQL query string is generated to insert data into the `INCIDENT` table in the database. The first portion of the query string contains the column names into which values are being inserted. The second portion of the query contains the actual data to be inserted. The values for this data are obtained from the Incident object on which this **insertIntoDB** method was called. Because of the one-to-many relationship between incidents and medications, incidents and allergies, and incidents and vitals measurements, each of these is stored in a separate table, with the incident number field serving as the foreign key to relate each record to its associated incident record.

To determine the correct incident number, the server application places a lock on the `INCIDENT` table so that any other insertions into the table are held. The insertion is then performed, and a subsequent query is performed to determine the maximum incident number that exists in the database after the insertion. This is taken to be the "current" incident number and is used in the related record insertions. Once the incident number has been determined, the `INCIDENT` table is

unlocked so that other transactions may take place. If the insertion queries are successful, the client is returned an “OK” status code, as described in section 4.3.7.

Function	public String insertIntoDB ()
Purpose	Inserts the data contained in the Incident object into the MySQL database
Parameters	None
Result	Returns the status string describing any errors that may have occurred during the operation. If any errors occur, an <code>SQLException</code> is thrown
Called By	This method is called from the Connection.run() method

4.3.6 Emailing Incident Reports

The Incident class’s **sendEmail** method makes use of Sun’s experimental API for interfacing with email servers, the `javax.mail` package. The first step in sending an email using the SMTP portion of this package is to create a Session object. This object manages the properties of the message, including transport type, message addressing information, and message body.

To create the message body, the **sendEmail** method calls the Incident object’s **toString** method, which creates a readable string representation of the object. The method then retrieves the to and from addresses from the Preferences object. If SMTP authentication has been enabled by the configuration manager, the SMTP transport will pass that information on to the SMTP server when sending the message.

If the SMTP transport detects that an error has been sent by the SMTP server, it will throw a `MessagingException`. In this case, the server application will print out the error to the console and continue servicing clients.

4.3.7 Providing Client Status

To provide the client with status as to whether the incident was successfully stored, the server sends a status code and string prior to the connection being closed. To facilitate this, throughout the connection, a status integer and string are maintained. If any exceptions are caught, this status is updated to reflect the event that occurred. For instance, if an `SQLException` is caught by the Connection object, the status code is changed to indicate that an SQL error has occurred, and the

status string is changed to that of the exception. The format of the server's response to the client is as follows: `ErrorNum:ErrorString`. The error codes that are returned by the server are shown in Table 3.1.

4.3.8 Reporting

Two methods are available to facilitate information gathering by the EMS department. A Web interface allows quick viewing of incidents that is similar to the all-in-one presentation of the paper run sheets. Additionally, a Microsoft Access database may be used to provide customizable aggregate reporting.

4.3.8.1 Web Interface

The EMS system's Web interface was written in PHP, mainly due to its cross-platform compatibility and well-established methods for connecting to MySQL databases. Each PHP script begins by calling the `mysql_connect`

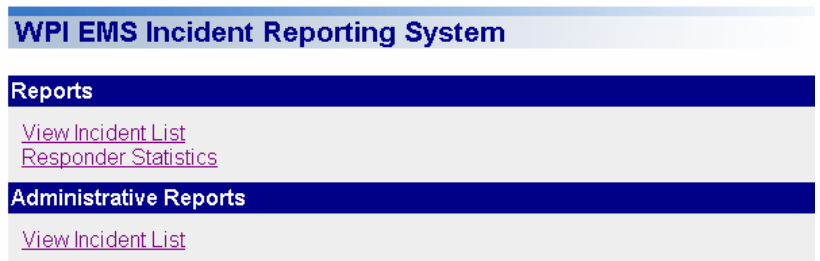


Figure 25 Web Interface Main Menu

method provided by PHP to connect to the EMS database. Depending upon the desired content of the page, an SQL query is formed and submitted to the database.

The results of the query are placed into a result set that can be processed row-by-row using an array metaphor. For instance, to print a listing of the incidents in the database, a `SELECT <attributes> FROM INCIDENT` query is run. For each row returned by the DBMS, the PHP script writes a line of HTML containing elements from the results.

There are two levels of access within the Web interface. The lower level provides access to incident information without any personally identifiable information visible. A person with higher-level permissions may access another tier of the Web interface which offers access to all of the incident

information. This access is controlled by the file permissions of the Web server – those pages which can access the personally identifiable information are protected by the Web server’s built-in security functions.

4.3.8.2 Microsoft Access

Because of its flexibility and user-friendly customization features, Microsoft Access was selected as the aggregate reporting tool. This will allow the users to create their own reports and customize the existing ones to their needs. A PHP-only solution would have restricted the changes that could be made and would have increased the technical knowledge required to implement those changes.

To facilitate the connection to the MySQL database, a MySQL ODBC connector must be loaded on the computer that is running Access. Once this connector has been installed, Access is able to view and manipulate the tables of the database. Through this ability, users can use Access’s report wizards to obtain the information they need. In addition to reporting capabilities, users may also use Access to administer the responders in the database, including performing password resets and adding or removing users.

5 Testing

5.1 Testing Methodology

A variety of testing techniques were employed to ensure proper functionality of the final system. The primary method of testing was the entry of a number of test cases that were designed to simulate real-world incidents. The execution of these tests served to test the entire system, including the client-side component, the transmission component, and the server-side component.

The secondary method of testing utilized “Gremlins,” a testing functionality built into the Palm OS Emulator. Gremlins perform automated actions designed to manipulate every user interface element in the Palm application. This portion of the testing only tested the client-side component.

The next phase of testing was to be field testing by EMS technicians. However, due to time constraints, this was not possible. In the future, the system will be field tested before deployment.

5.2 Results/Concerns

As a result of the test cases performed, a number of defects were found and repaired in the original implementation. Field testing must be conducted to ensure correct operation of the system. The system should not be deployed until this has occurred.

6 Future Research

While the EMS Portable Workflow System is operational, its design and implementation, along with the resources Palm OS 5 provides to its users, leaves many options for future work topics. The existing system could benefit from upgrades such as creating a conduit that allows for synchronization through a serial connection to a PC, allowing the patient to be able to sign and refuse treatment, and the ability to have technicians include images and sound files with each incident report.

6.1 Conduit Development

Development of a conduit for the EMS Portable Workflow system would enable data to be sent and received through the HotSync manager, which is invoked when the user presses the button on the cradle the device sits in. The cradle could be connected to any PC that has Internet access through a Universal Serial Bus (USB) port cable. It does not necessarily have to be connected to the machine that the EMS database resides on. With a conduit designed, it would also be possible to perform the HotSync operation over the wireless network.

Enabling the user to synchronize data by way of the HotSync operation would be beneficial if the wireless network is ever unavailable for some reason. There would be no extra hardware needed for this project to be completed. The device comes equipped with both the cable and cradle it needs to perform the operation.

If a conduit were to be developed for this system, the SSL encryption technique should be applied to the network transmission code. A request to WPI's Technical Services department could be made to develop an outgoing SMTP mail server which supports SSL functionality. WEP would still be enabled to ensure security over the wireless medium, but data that is sent over the wired medium would now be secure.

6.2 Adding Signature Functionality

The EMS technician will still need to carry around a small piece of paper in the event a patient wishes to refuse treatment. The incident report will still be filed, but no treatment is given to that patient. For a patient to refuse treatment, they must sign a piece of paper to verify that they are in fact refusing the treatment.

The EMS Portable Workflow System could be enhanced to provide the ability to have a patient enter their signature on the device and have it be stored with the corresponding incident report. Associating each signature with the appropriate incident report would be a formidable task. Likewise, creating a canvas on which the patient could sign would most likely need to be created within the existing application, and could prove to be a difficult but rewarding task.

Careful considerations would need to be made in how the signature would be stored and represented on both the handheld device and server application. The most obvious way would be to store the signature as an image file.

6.3 Storing/Sending Multimedia

As PDA technology continues to grow, the ability to support more and more multimedia will be incorporated into each device. Devices are able to store image and audio files on a memory card that is inserted in the memory card slot, the device itself. The EMS Portable Workflow System could benefit significantly if multimedia assets were to be incorporated into its system.

Some EMS groups take pictures of incidents for their records. The Sony Clié series as well as other manufacturers of PDA's, design models which have the ability to take digital pictures. To be able to send image files to the server, the appropriate device (one that is able to take digital pictures) would need to be purchased. A memory card would be needed as well to allow for multiple images to be stored.

The storage and sending of audio files would allow the technician to quickly take notes of a particular incident and send it along with a corresponding incident report. Just as in the case of including image files, if the ability to store many audio files on the device was desired, a memory card would need to be purchased for the device.

In both the image and audio file scenarios, there would need to be an association made to have the appropriate image/audio files correspond to the correct incident report. There could be multiple images/audio files sent along with a single report. The association of the multimedia files from the databases they are stored in on the device would require a good deal of work, but if completed successfully, would add much more power to the system.

7 Conclusion

The goal of this project was to streamline the incident reporting process for the WPI EMS department to lower administrative overhead. The EMS Portable Workflow system successfully implemented a design based on the client-server model, utilizing personal digital assistant technology to implement the client-side component. Data is transported from clients to the server using wireless technology. The server receives the data from the network and inserts it into a database management system.

Working with such new technologies presented interesting challenges during the execution of the project, due to the limited amount of knowledge available. Overcoming these challenges required innovative techniques to be employed, as well as modifications to the scope of the project. Another major challenge stemmed from the process of connecting such a variety of diverse technologies.

Personal digital assistant and wireless transmission technologies continue to grow. The system is adaptable to newer developments as they emerge, due to the use of platform independent technologies. It is the hope of the authors that the EMS Portable Workflow System provides long and

useful service to the WPI EMS department and may someday be expanded to serve other EMS departments.

8 References

- BACH02 Bachmann, Glenn. A Survey of Wireless Options for Palm Developers. *Earthweb Networking and Communications*. Accessed October 26, 2002.
http://softwaredev.earthweb.com/wireless/article/0,,10836_1480501_2,00.html
- BANE00 Banerjee, Sandeepan, Vishu Krishnamurthy, Muralidhar Krishnaprasad and Ravi Murthy. (2000, February 28-March 16). "Oracle 8i – The XML Enabled Data Management System." Proceedings, 16th International Conference on Data Engineering. San Diego, CA. IEEE, 2000. 561-569.
- CHIP00 Chipman, Mary, Andy Baron, Chris Bell, Michael Kaplan, Paul Litwin, and Rudy Torrico. (2000, October). "Frequently Asked Questions About Microsoft Access Security for Microsoft Access versions 2.0 through 2000." Microsoft Corporation.
<http://support.microsoft.com/support/access/content/secfaq.asp>
- FOST02 Foster, Lonnon R., (2002) Palm OS Programming Bible, Second Edition, Indianapolis, IN: Wiley Publishing, Inc.
- GAMM95 Erich Gamma et. Al. (1995). *Design Patterns: Elements of Reusable Software*, Addison Wesley: Boston, MA.
- GEIE02 Geier, Jim. (June 20, 2002) "802.11 WEP: Concepts and Vulnerability." 802.11-planet.com. <http://www.80211-planet.com/tutorials/article.php/1368661>
- GREE01 Greenwald, Rick, Robert Stackowiak and Jonathan Stern. (2001). *Oracle Essentials: Oracle9i, Oracle8i, & Oracle8, Second Edition*. Sebastopol, CA: O'Reilly & Associates.
- GRIM98 Grimes, Seth. (1998, April). "Modeling Object/Relational Databases." *DBMS*.
<http://www.dbmsmag.com/9804d13.html>
- HEDR87 Hedrick, Charles L. "General Description of the TCP/IP Protocols." (1987).
<http://oac3.hsc.uth.tmc.edu/staff/snewton/tcp-tutorial/sec2.html>
- INFO02 InfoPlease.com. (2002), "Meaning of Pixel."
<http://www.infoplease.com/ipd/A0590740.html>
- KIBB01a Kibbe, David C., MD. (2001, July/August). "A Problem-Oriented Approach to the HIPAA Security Standards." *Family Practice Management*. pp. 37+.
- KIBB01b Kibbe, David C., MD. (2001, March). "What You Need to Know About HIPAA Now." *Family Practice Management*. pp. 43+.
- KING99 King, Tim, George Reese, and Randy Jay Yarger. (1999). *MySQL & mSQL*. Sebastopol, CA: O'Reilly & Associates.
- MCCL97 McClure, Steve. (1997, August). "Object Database vs. Object-Relational Databases." *IDC Bulletin #14821E*. <http://www.cai.com/products/jasmine/analyst/idc/14821E.htm>

- MORA02 Moran, Joseph. (2002, November 6). "Wireless Home Networking, Part III - Wi-Fi Security." 802.11-planet.com.
<http://www.80211-planet.com/tutorials/article.php/1495811>
- MUEN02 Muench, Steve. (2002). *Building Oracle XML Applications*. Sebastopol, CA: O'Reilly & Associates.
- PALM03a "Conduit Development." (2003). <http://www.palmos.com/dev/tech/conduits/>
- PALM03b "Palm OS 5 Development Overview." (2003).
<http://www.palmos.com/dev/support/docs/palmos50/>
- PALM03c "Palm OS Application Design." (2003).
http://www.palmos.com/dev/support/docs/ui/UI_Design.html
- PALM03d "Palm OS." (2003). <http://www.palmsource.com/palmos/>
- PALM03e "Palm Ethernet Cradle FAQ." (2003)
http://www.palm.com/support/enterprise/ecradle_faq.html
- PALM03f "Palm User Interface Guidelines." (2003).
<http://www.palmos.com/dev/support/docs/ui/UIGuidelinesTOC.html>
- PALM03g "Palm API Reference." (2003).
<http://www.palmos.com/dev/support/docs/palmos/ReferenceTOC.html>
- PALM03h "PDB and PRC Database Formats." (2003).
http://www.palmos.com/dev/support/docs/fileformats/PDB_PRCFormat.html#939350
- SMIT01 Smith, Megan. (2001, November). "TEMS celebrates 20 years of service." *Tulane Hullabaloo*. <http://hullabaloo.tulane.org/index.php/20011130/news/467/>
- STAN02 "Standards for Privacy of Individually Identifiable Health Information (Unofficial version)." 45 CFR Parts 160 and 164. Aug. 14, 2002.
<http://www.hhs.gov/ocr/combinedregtext.pdf>
- STRE02 Streger, Matthew R., MPA, NREMT-P. (2002, August). "It's All in the Palm of Your Hand." *Emergency Medical Service* (31), 8. pp. 77+.
- SONY03 "Sony Style – Sony Clié NX60." (2003).
http://www.sonystyle.com/is-bin/INTERSHOP.enfinity/eCS/Store/en/-/USD/SY_DisplayProductInformation-Start;sid=1oSc0hPjqKcfnYYaMWWeAdDyPJUPgZXSoQ=?CatalogCategoryID=&ProductID=M0MKC0%2eNcYMAAADzXZxypn_9&Dept=hp
- TANE03 Tanenbaum, Andrew S. (2003) *Computer Networks Fourth Edition*. Prentice Hall. Upper Saddle River, NJ.
- ULLM97 Ullman, Jeffrey D., and Jennifer Widom. (1997). *A First Course in Database Systems*. Upper Saddle River, NJ: Simon & Schuster.
- VOTS02 Votsch, Victor, and Mark Walter. (2002, March). "Oracle XML DB: Uniting XML Content and Data." Media, PA: Seybold Consulting Group.

- WIFI02 Wi-Fi Alliance. (2002). "Securing Your Wi-Fi Network." Wi-Fi Alliance.
http://www.wirelessethernet.org/opensslsection/secure_the_network_setup.asp
- WINT01 Winton, Greg. (2001). *Palm OS Network Programming*. Sebastopol, CA: O'Reilly & Associates.

9 Appendix

9.1 Incident Reports

9.1.1 Norwood Call Sheet

Norwood Emergency Medical Services Basic Life Support Report

Report No _____

Patients Name _____ Male Female Age _____ Est _____ Weight _____ DOB _____

Incident Location _____

Patients HOME Address _____

STATUS ON ARRIVAL <input type="checkbox"/> Alert <input type="checkbox"/> Verbal <input type="checkbox"/> Pain <input type="checkbox"/> Confused <input type="checkbox"/> Pt. Denies Complaint <input type="checkbox"/> Pain (Chest) <input type="checkbox"/> Syncope <input type="checkbox"/> Near Syncope <input type="checkbox"/> Respiratory <input type="checkbox"/> Combative <input type="checkbox"/> Unconscious <input type="checkbox"/> Arrest <input type="checkbox"/> Pain from Injury <input type="checkbox"/> Pain Abdomen <input type="checkbox"/> Seizure <input type="checkbox"/> Other _____																																																																																	
CPR by <input type="checkbox"/> PD <input type="checkbox"/> BLS <input type="checkbox"/> ALS <input type="checkbox"/> BS Est Anoxic Time _____		Patient How / Where Found _____		Patients Physician _____ Hospital _____																																																																													
HPI _____ Symptoms w/Onset _____		<input type="checkbox"/> Angina <input type="checkbox"/> CHF <input type="checkbox"/> CVA <input type="checkbox"/> COPD <input type="checkbox"/> MI <input type="checkbox"/> Hypertension <input type="checkbox"/> Pacemaker <input type="checkbox"/> Diabetes <input type="checkbox"/> CA of _____ <input type="checkbox"/> Other _____		<input type="checkbox"/> Distress <input type="checkbox"/> Yes <input type="checkbox"/> No <input type="checkbox"/> Mild <input type="checkbox"/> Moderate <input type="checkbox"/> Severe <input type="checkbox"/> Sudden Onset <input type="checkbox"/> Gradual Onset <input type="checkbox"/> Apneic <input type="checkbox"/> Airway Patent																																																																													
Precipitating Events _____ Responds To <input type="checkbox"/> Alert <input type="checkbox"/> Verbal <input type="checkbox"/> Pain <input type="checkbox"/> Unresponsive CC-Suspected Injuries _____		Condition <input type="checkbox"/> Warm <input type="checkbox"/> Pink <input type="checkbox"/> Dry <input type="checkbox"/> Pale <input type="checkbox"/> Cool <input type="checkbox"/> Flushed <input type="checkbox"/> Hot <input type="checkbox"/> Cyanotic <input type="checkbox"/> Moist <input type="checkbox"/> Ashen <input type="checkbox"/> Normal	Color <input type="checkbox"/> Pink <input type="checkbox"/> Pale <input type="checkbox"/> Flushed <input type="checkbox"/> Cyanotic <input type="checkbox"/> Ashen <input type="checkbox"/> Normal	Rt <input type="checkbox"/> Lt <input type="checkbox"/> <input type="checkbox"/> PERL <input type="checkbox"/> _____ <input type="checkbox"/> Normal <input type="checkbox"/> Delayed																																																																													
<table border="1"> <thead> <tr> <th>TIME</th> <th>PULSE</th> <th>B/P</th> <th>RESP</th> <th>TREATMENT</th> <th>BY</th> </tr> </thead> <tbody> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td></tr> </tbody> </table>		TIME	PULSE	B/P	RESP	TREATMENT	BY																																																																									Incident Commander _____ EMS Branch Director _____ Triage _____ Treatment _____ Transport _____	
TIME	PULSE	B/P	RESP	TREATMENT	BY																																																																												
Response to Treatment _____		Unit # _____ Disp by _____ Released by _____ <input type="checkbox"/> NUA Arrived <input type="checkbox"/> prior to BLS <input type="checkbox"/> with BLS <input type="checkbox"/> after BLS		_____																																																																													
Narrative _____ _____ _____ _____ _____																																																																																	
Date _____ Dispatched _____ (AM / PM) Responding _____ (AM / PM) On Scene _____ (AM / PM) Departed _____ (AM / PM) At Hosp _____ (AM / PM) Available _____ (AM / PM) OOS/HQ _____ (AM / PM)		Required Attachments Defibrillation - Attach EMT-D Form and Code Summary Multiple Patient Incident - Attach ICS 201 PRIMARY Crew & Vehicle Ambulance No _____ C _____ R _____ E _____ W _____ Driver _____		Disposition <input type="checkbox"/> Treated <input type="checkbox"/> Refused EMS Transported To <input type="checkbox"/> Hospital _____ <input type="checkbox"/> Left at Scene <input type="checkbox"/> Home Transferred Care (Time) <input type="checkbox"/> Airmedical <input type="checkbox"/> Emergency Rm Staff _____ <input type="checkbox"/> Other EMS Service _____																																																																													
Documented By _____		QA Review By _____																																																																															

9.1.2 WPI's Existing Call Sheet

Worcester Polytechnic Institute Emergency Medical Services:

Date: ____/____/____ Patient Priority/Distress: 1-Severe 2-Moderate 3-Mild Dispatch Location: _____

Patient Name: _____ WPI ID: _____ - _____ - _____ DOB: ____/____/____

Sex: M F

Age: _____ WPI Box: _____ Dorm Room: _____ Address (if not WPI): _____

Mechanism of Injury: _____

History/Narrative on present illness/injury (continue on back if necessary): _____

Check here if continued on back: []

Past Pertinent History:

Medications:

1 _____ 2 _____ 3 _____ 4 _____ 5 _____

Allergies: [] NKDA

1 _____ 2 _____ 3 _____ 4 _____ 5 _____

Vital Signs: [] Unable to obtain [] Not attempted

Time	BP	HR	RR

HR: [] Regular []

Weak [] Bounding

[] Thready [] Irregular

RR: [] Clear [] Wheezing [] Labored [] Shallow [] Absent

Skin: [] Hot [] Warm [] Cool [] Cold [] Pale [] Flushed [] Cyanotic [] Normal [] Dry

Pupils: R: [] PEARL [] Unreactive [] Dilated [] Constricted [] Slow

L: [] PEARL [] Unreactive [] Dilated [] Constricted [] Slow

Emergency Care: [] Oxygen: _____ LPM via: NRB NC BVM Blow-by

[] Suction [] Bandaging [] Spinal Immobilization

[] Ventilation [] Ice / Heat Packs (circle one) [] CPR

[] Airway insertion [] Splinting [] Psychological

[] Other: _____

Times:

Call: _____ Dispatch : _____ On scene: _____ Clear scene: _____ Est. total time: _____

EA called: _____ EA on scene: _____ EA clear: _____ Transported by: _____ Transported to: _____

WPI EMS:

Name: _____ Signature: _____ Number: _____ WPI Police On Scene: _____ Officer Name: _____

Number: _____

Name: _____ Signature: _____ Number: _____ Officer Name: _____

Number: _____

I acknowledge that I was offered medical assistance by the Worcester Polytechnic Institute Emergency Medical Services and that I refuse that care. I understand that I can call WPI EMS at any time in the future if I choose to do so.

Patient Signature: _____ Witness Signature: _____

WPI EMS Attendant Signature: _____

Prehospital Care Report faxed to EMS Coordinator: _____: _____ AM / PM

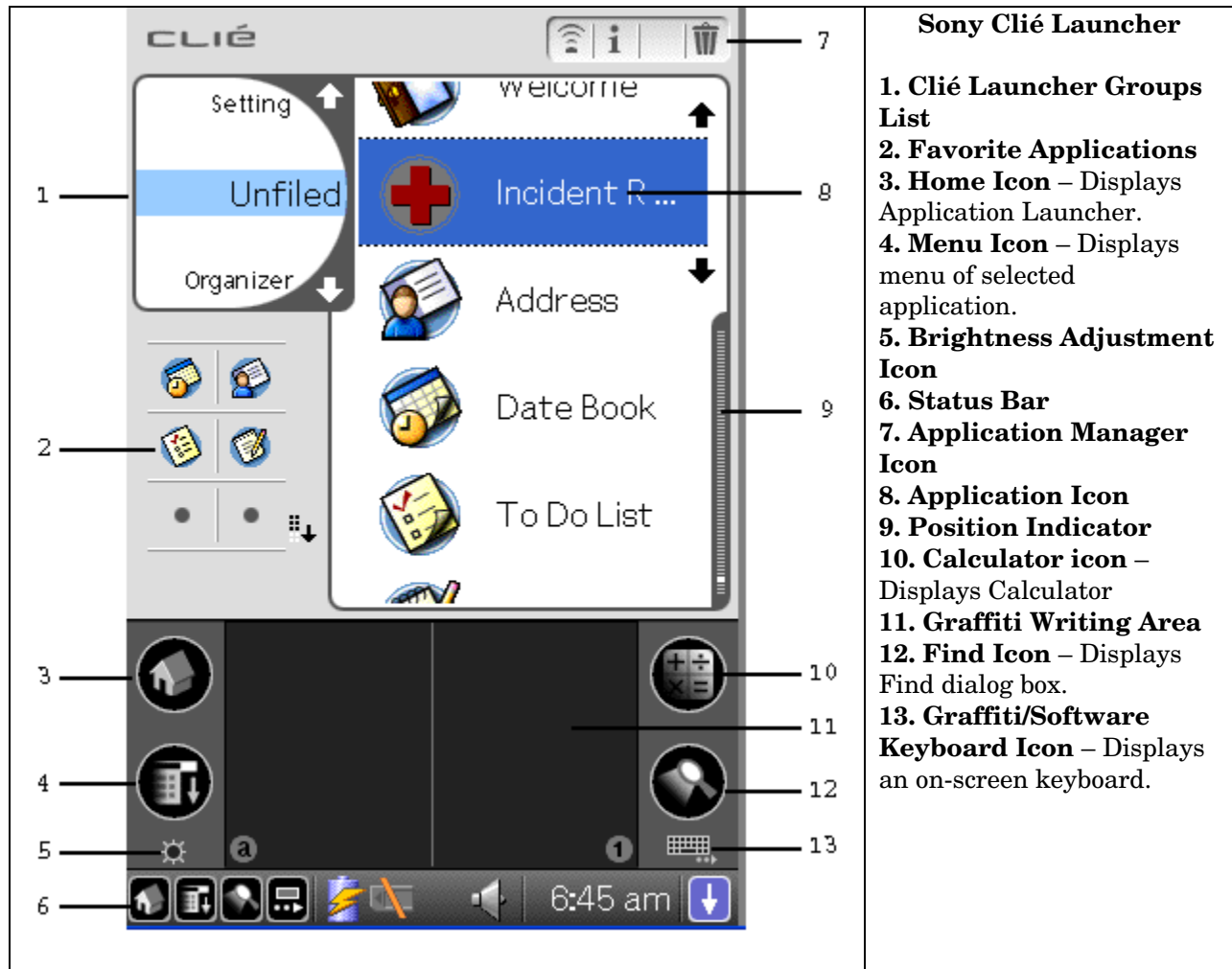
9.2 Sony Clié NX60

9.2.1 NX60



(SONY03)

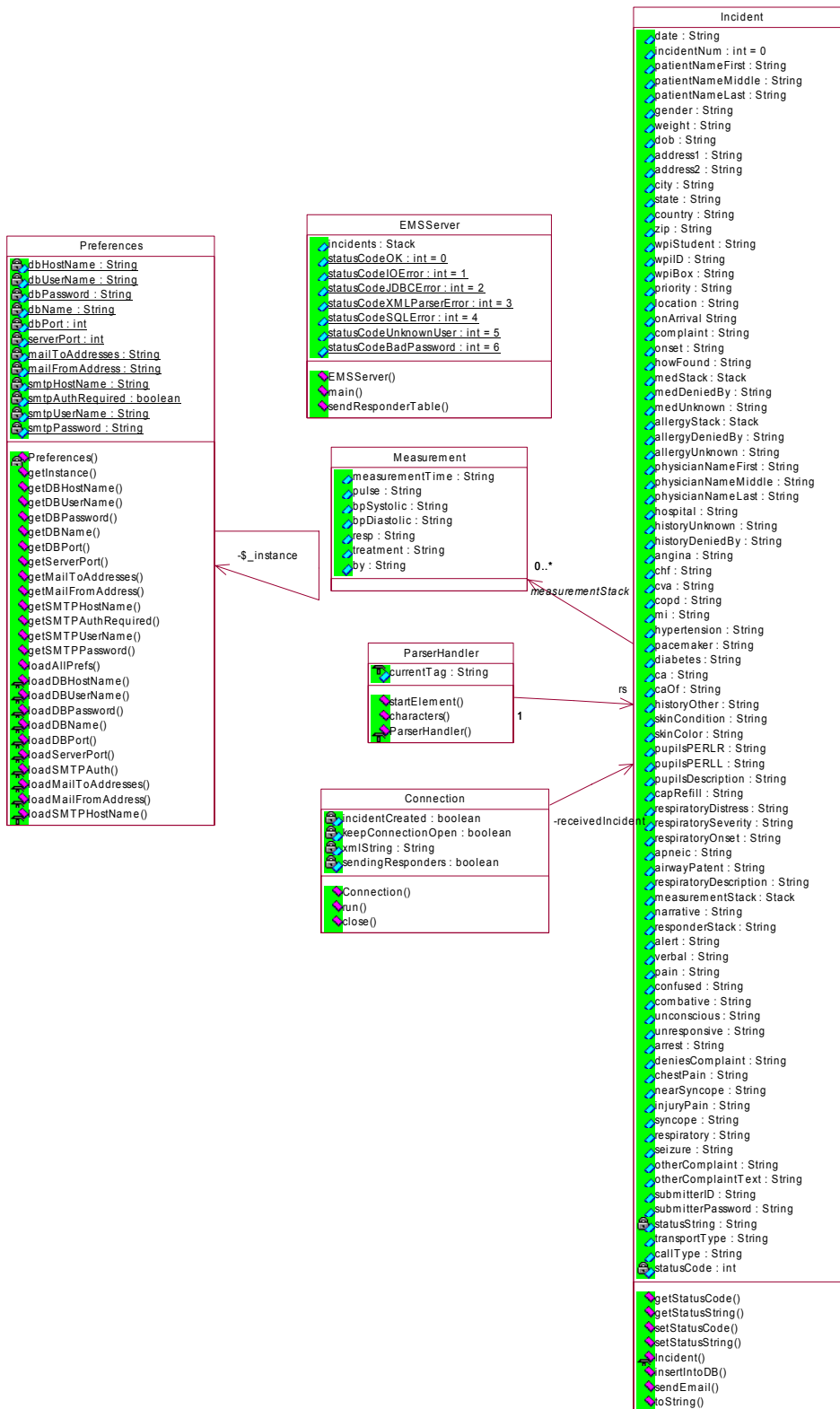
9.2.2 Sony Clié Launcher



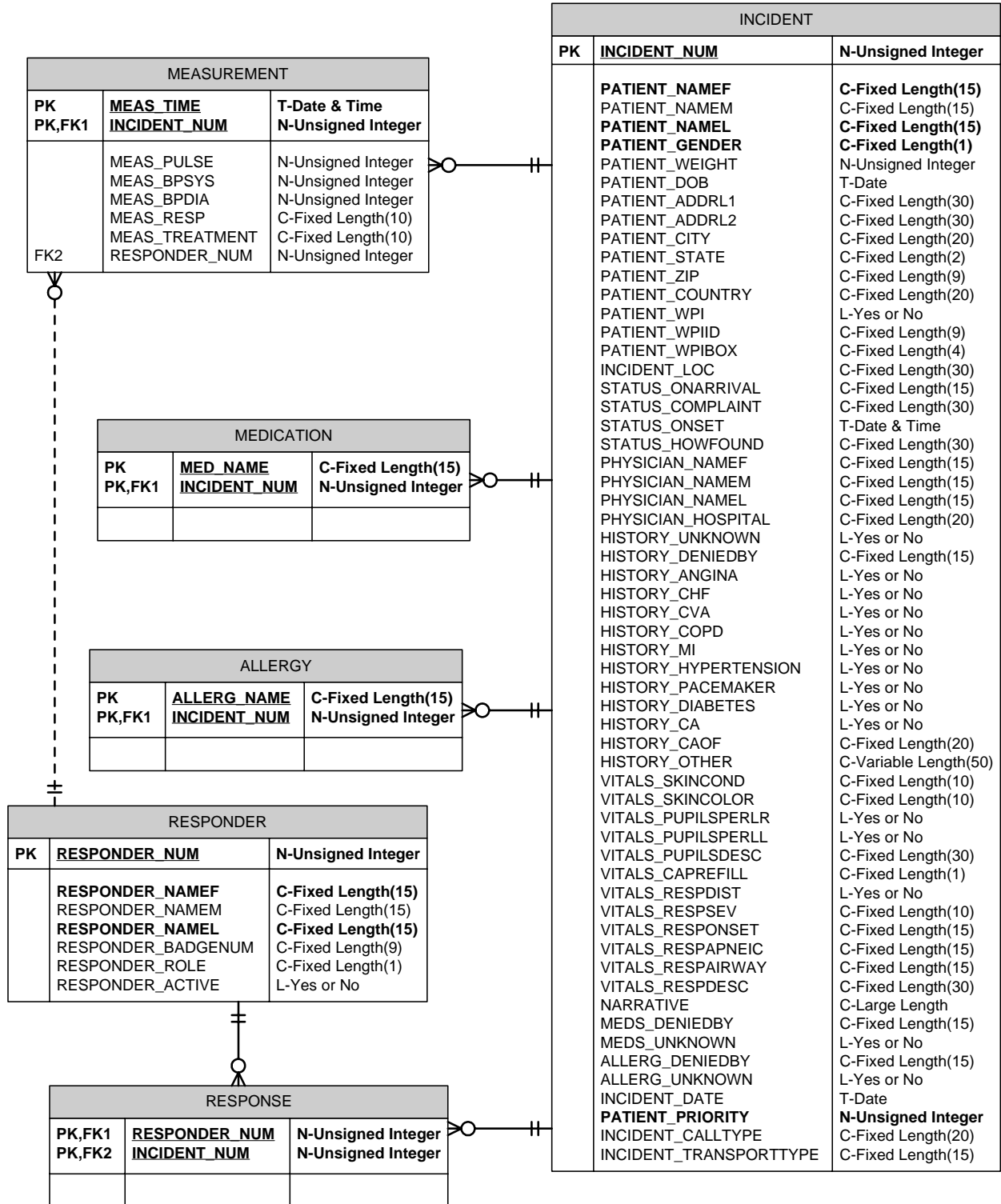
Sony Clié Launcher

1. Clié Launcher Groups List
2. Favorite Applications
3. Home Icon – Displays Application Launcher.
4. Menu Icon – Displays menu of selected application.
5. Brightness Adjustment Icon
6. Status Bar
7. Application Manager Icon
8. Application Icon
9. Position Indicator
10. Calculator icon – Displays Calculator
11. Graffiti Writing Area
12. Find Icon – Displays Find dialog box.
13. Graffiti/Software Keyboard Icon – Displays an on-screen keyboard.

9.3 Server Design Information



Data Design

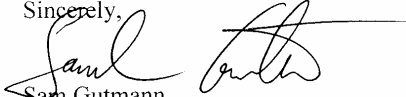


9.4 EMS Acceptance Letter

To whom it may concern:

On behalf of the Worcester Polytechnic Institute Emergency Medical Services department, I would like to thank the EMS Portable Workflow MQP group for their work in creating a system that will greatly ease our daily workload.

I hereby formally recognize this project as meeting our original specifications.

Sincerely,

Sam Gutmann

9.5 Users' Manual

For OS 5 and device specific problems, please refer to the “Read This First – Operating Instructions” booklet supplied with the device. You may also refer to the official Sony Clié Web site at <http://www.sony.com/clié/>

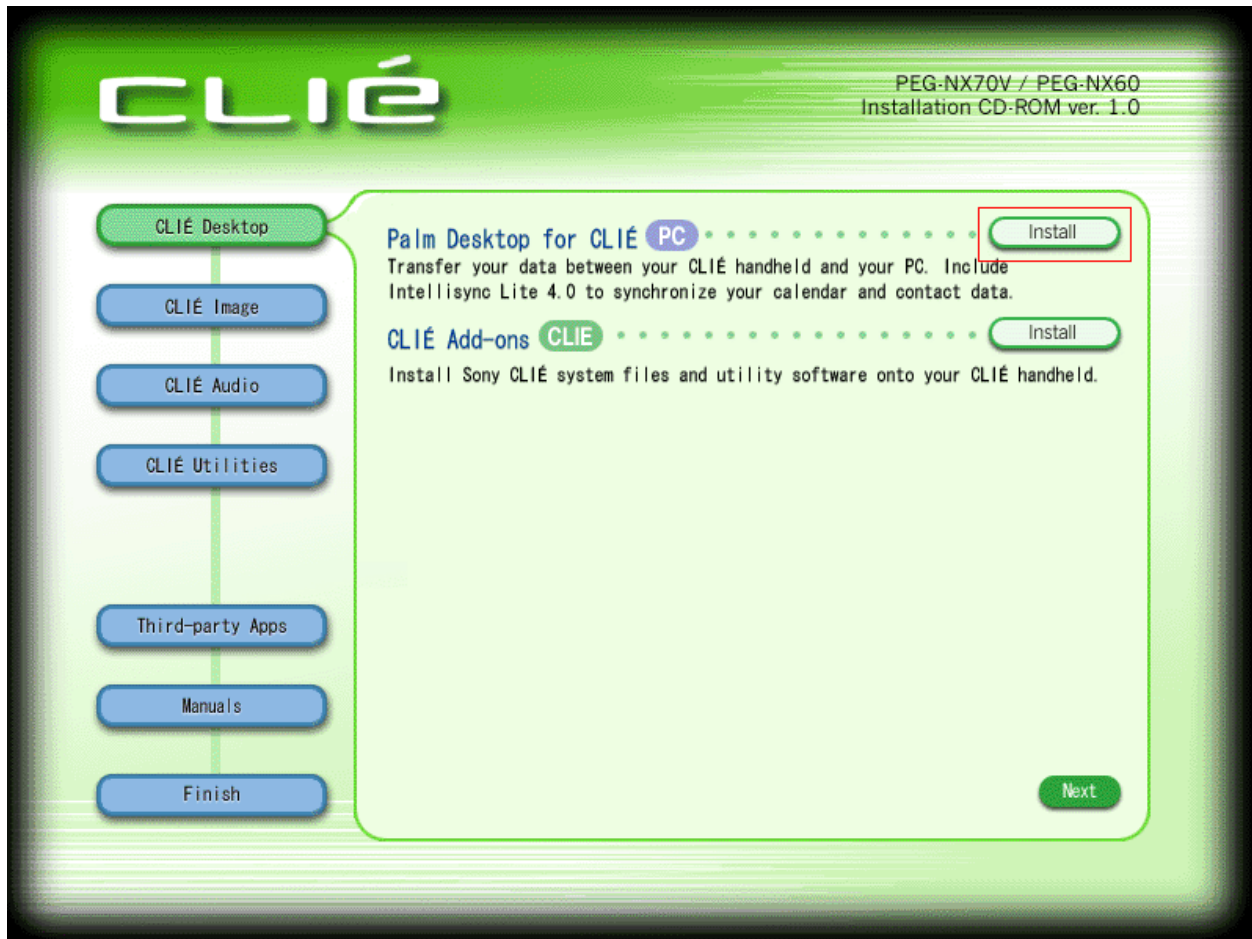
9.5.1 Installing Application Software

For general questions on how to use the Sony Clié software, please refer to the Sony “Read This First – Operating Instructions” manual that comes with the device.

9.5.1.1 Installing Sony Clié Desktop Software

To install the Emergency Incident Reporting Application on the Sony Clié, Sony Clié Palm OS specific software must be installed on a PC to which the NX60's Hotsync cradle is attached.

Insert the Sony Clié Installation CD-ROM Version 1.0 for PEG-NX70V/U and NX60/U into your CD-ROM drive. The following menu will appear:

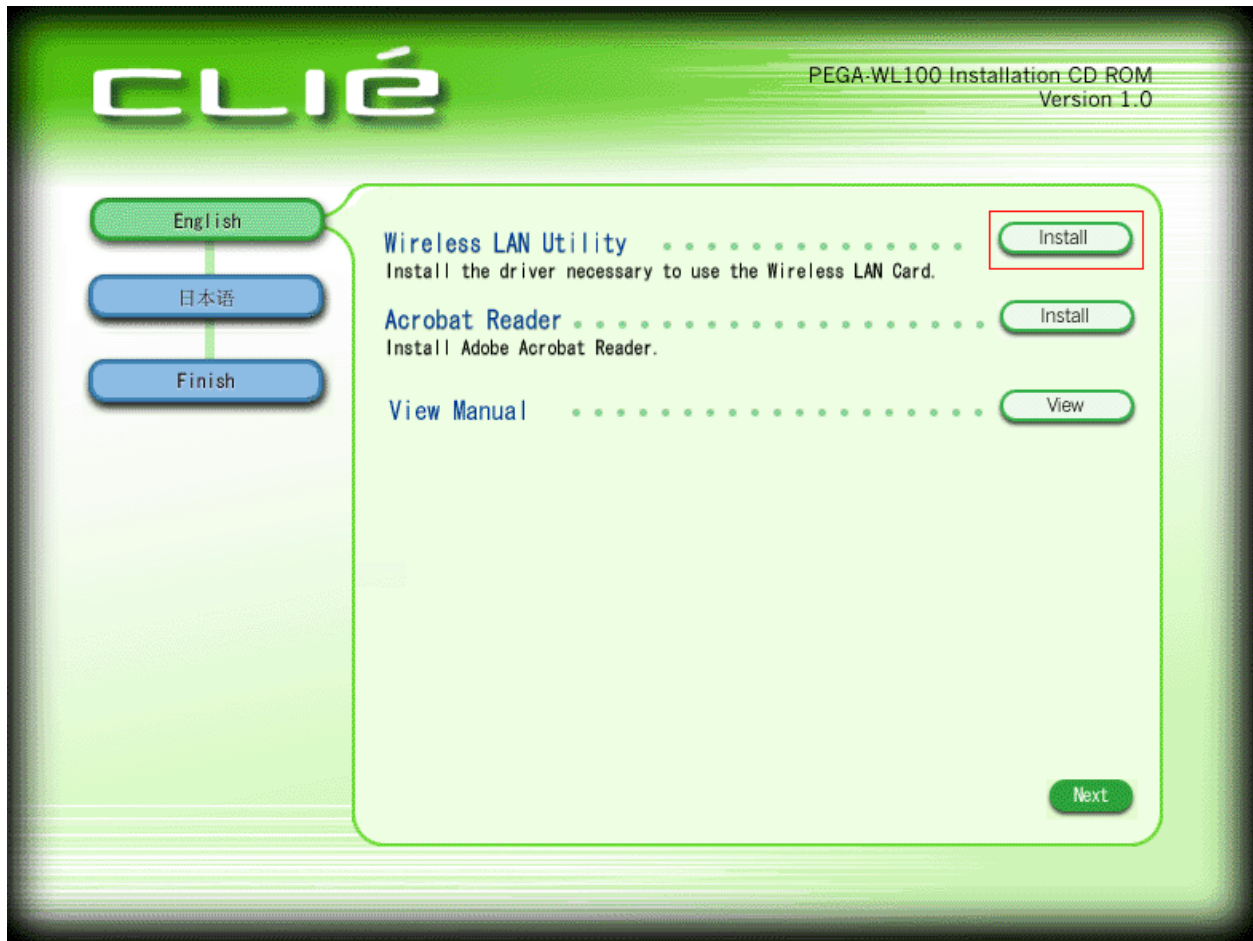


Click on the Install button highlighted in **RED** to install the Clié desktop software on your machine.

9.5.1.2 Installing Wireless Card Software

For the PEGA-WL100 to work correctly with the Sony Clié NX60, Wireless Card Software must be installed to the device.

Insert the Sony Clié Installation CD-ROM Version 1.0 for Wireless LAN Card PEGA-WL100 into your CD-ROM drive. The following menu will appear:

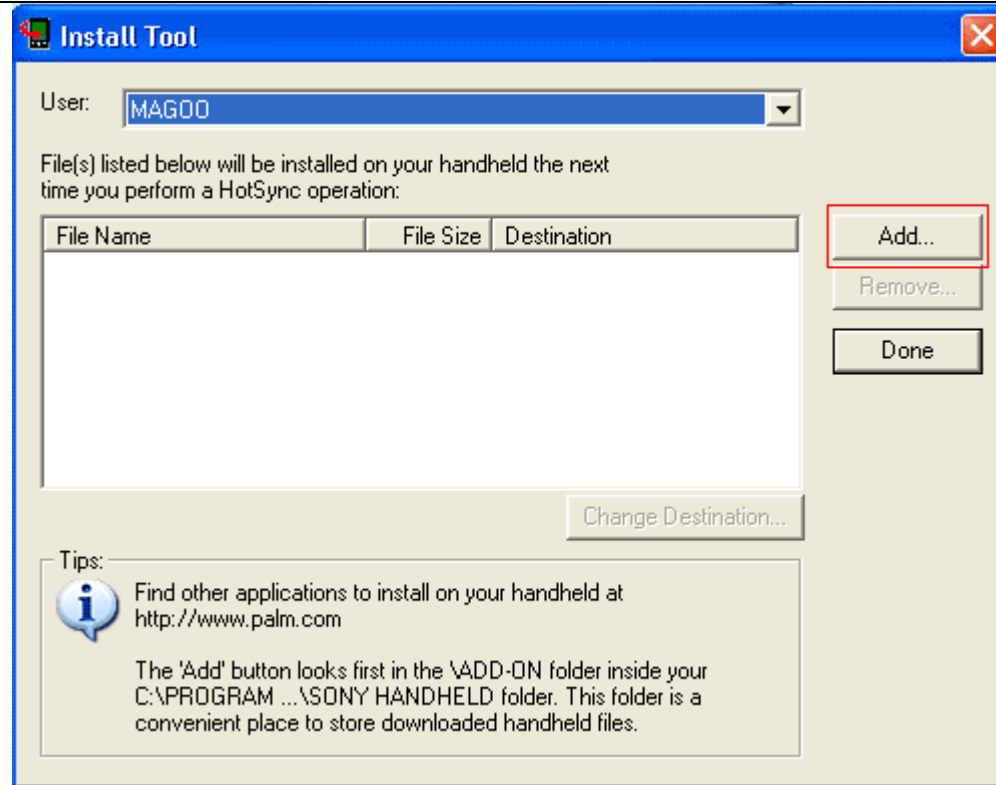


Click on the Install button highlighted in **RED** to install the Clié Wireless Card software on your machine, which will eventually be installed on the device.

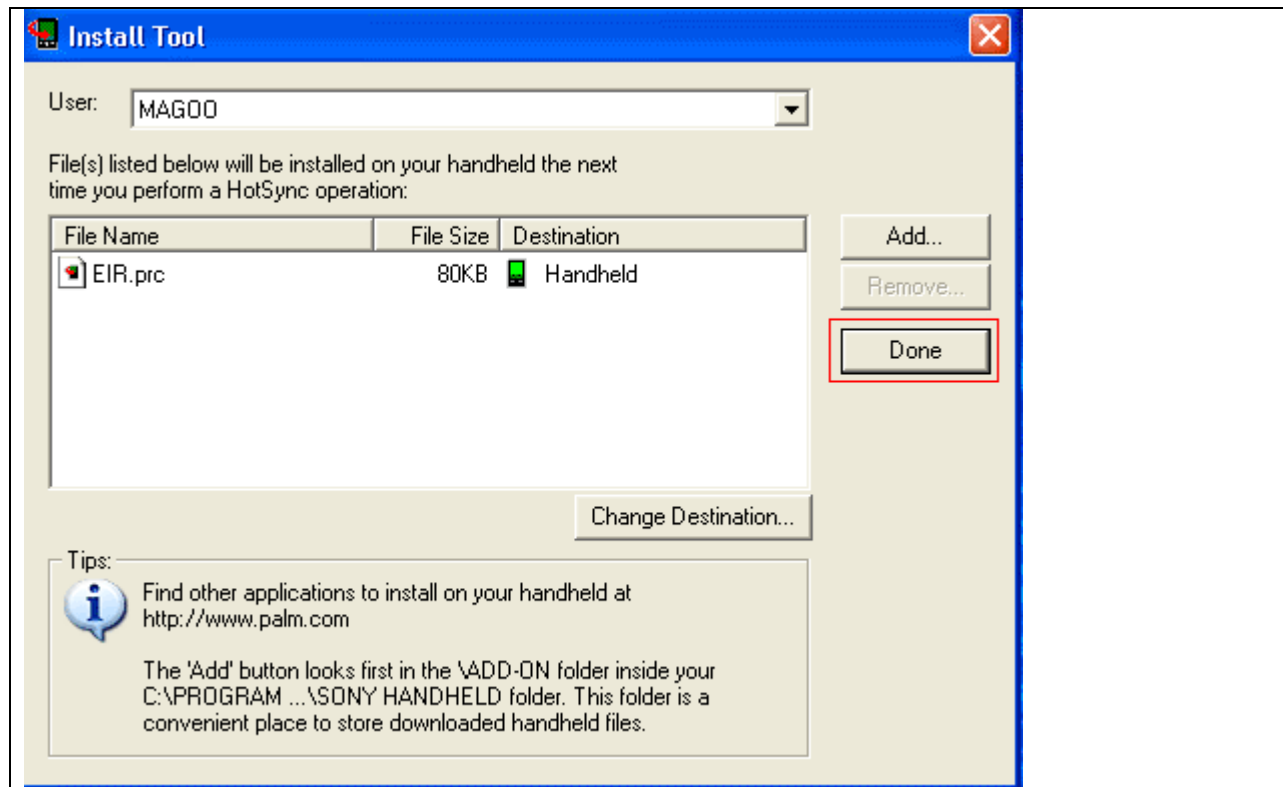
9.5.1.3 Installing the Emergency Incident Reporting Application

Use the following instructions to install the Emergency Incident Reporting application EIR.prc.

1. Insert the EMS Portable Workflow CD-ROM into your CD-ROM drive.
2. From the Start Menu, locate and open the Install Tool from the Sony Clié program folder. The following screen will appear.




3. Click the Add... button to add a new application to the device.
4. Navigate to the EIR.prc file on the CD-ROM disc, and Open it.
5. EIR.prc will now appear in the window on the left.




6. Select Done when the EIR.prc has been entered into the list on the left.
7. The next time you perform a Hotsync operation, the Emergency Incident Reporting Application, EIR.prc will be installed to the Sony Clié NX60.

9.5.2 Using the EMS Application on the Handheld


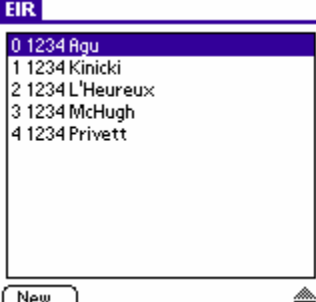


Sony Clie Launcher



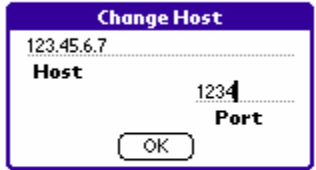
1. Open the Emergency Incident Reporting application by tapping the  icon.

9.5.2.1 Emergency Incident Reporting Main Screen

9.5.2.1.1 Main Menu

	<p style="text-align: center;">Main Menu</p> <ol style="list-style-type: none"> 1. Tap the EIR tab to access the Application Menu Bar. 2. Tap the ▲ or ▼ to scroll through Incident Reports.
	<p style="text-align: center;">Incident Report List</p> <ol style="list-style-type: none"> 1. Reports are numbered in the order they were created. This number appears to the far left of each report listing. 2. The number after that is the Responder ID. 3. The final part of each Incident Report listing is the Last Name of the patient. 4. Tapping on a report will take you to the Patient Information form for that report.

9.5.2.1.1.1 Setting Host Name and Port Number

	<p style="text-align: center;">Menu Bar for the Application Main Menu</p> <ol style="list-style-type: none"> 1. Access this menu by tapping the EIR tab, or by tapping the  icon. 2. Set the Host and Port information by tapping the Set Server... 3. Tapping About displays information about the application.
	<p style="text-align: center;">Changing the Server Information</p> <ol style="list-style-type: none"> 1. Change the Host by entering the Host of the Server you which to connect to. This may either be the actual server name, or an IP address. 2. Set the Port by entering the Port Number of the Server on the Host machine.

9.5.2.1.1.2 Retrieving Responder List

<p>The screenshot shows the application menu bar with 'File', 'About', 'Set Server...', and 'Update Responders...' options. A 'New...' button is visible at the bottom left.</p>	<h4 style="text-align: center;">Retrieving Responder List</h4> <ol style="list-style-type: none"> 1. To retrieve a new list of Responders, tap Update Responders... 2. A menu will appear informing you that the application is connecting to the network. 3. A message will appear detailing the status of the transfer.
--	---

9.5.2.2 Incident Report Forms

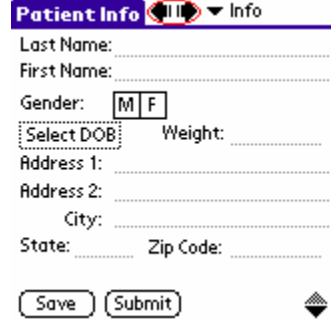
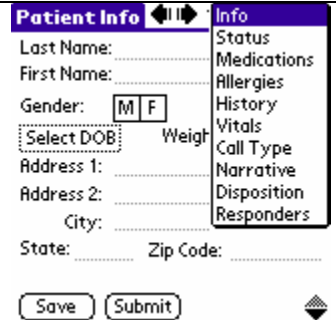
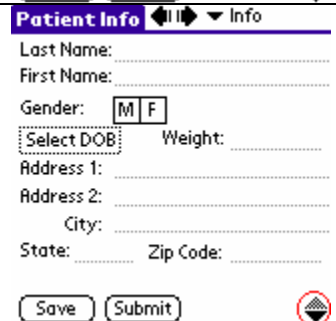
9.5.2.2.1 Incident Report Menu Bar

The Application Menu Bar is the same for all Incident Report forms.

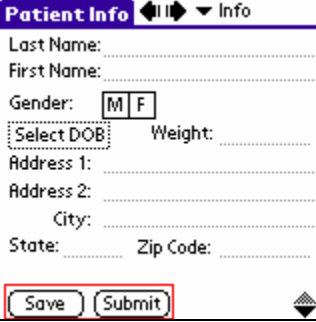

<p>The screenshot shows the Incident Report form with the 'File' menu open. The 'Delete...' option is highlighted, indicating the user is about to delete the current report.</p>	<h4 style="text-align: center;">Incident Report Menu Bar File Option</h4> <ol style="list-style-type: none"> 1. From the File option, you can delete the incident report on which you are currently working. This is the only point from which you can do this. 2. Selecting Delete will delete the incident report and take you back to the main menu.
<p>The screenshot shows the Incident Report form with the 'File' menu open. The 'Edit' menu is expanded, showing options like Undo, Cut, Copy, Paste, Select All, Keyboard, and Graffiti Help.</p>	<h4 style="text-align: center;">Incident Report Menu Bar Edit Option</h4> <ol style="list-style-type: none"> 1. The Edit Menu options allow for basic Editing functionality. 2. Basic functionality includes undo, cut, copy, paste, and select all. 3. Selecting Keyboard displays an on-screen keyboard. 4. Selecting Graffiti Help displays the standard Graffiti keystroke reference list.

9.5.2.2.2 Incident Report Form Navigation

The navigation between forms is the same for each form in the Incident Report.



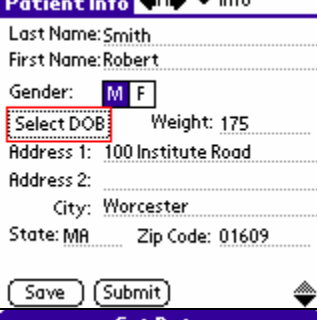

 <p>Patient Info ◀▶▼ Info</p> <p>Last Name: _____ First Name: _____ Gender: <input type="checkbox"/> M <input type="checkbox"/> F Select DOB: _____ Weight: _____ Address 1: _____ Address 2: _____ City: _____ State: _____ Zip Code: _____</p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/></p>	<h4 style="text-align: center;">Incident Report Form Navigation Arrows</h4> <ol style="list-style-type: none"> 1. Tap the ◀ and ▶ arrows to take you to the previous or next form respectively.
 <p>Patient Info ◀▶▼ Info</p> <p>Last Name: _____ First Name: _____ Gender: <input type="checkbox"/> M <input type="checkbox"/> F Select DOB: _____ Weight: _____ Address 1: _____ Address 2: _____ City: _____ State: _____ Zip Code: _____</p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/></p> <p style="text-align: right;"> Info Status Medications Allergies History Vitals Call Type Narrative Disposition Responders </p>	<h4 style="text-align: center;">Incident Report Navigation Menu</h4> <ol style="list-style-type: none"> 1. The Incident Report Navigation Menu provides fast access to any form. 2. To access the menu, tap on the form listing in the upper-right-hand corner of the screen. 3. Tapping on the appropriate form takes you to that form.
 <p>Patient Info ◀▶▼ Info</p> <p>Last Name: _____ First Name: _____ Gender: <input type="checkbox"/> M <input type="checkbox"/> F Select DOB: _____ Weight: _____ Address 1: _____ Address 2: _____ City: _____ State: _____ Zip Code: _____</p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/></p>	<h4 style="text-align: center;">Incident Report Navigation Scrollbars</h4> <ol style="list-style-type: none"> 1. Since the Palm Screen is small, you may need to scroll to get to areas of certain forms. 2. Tap the ▲ or ▼ to scroll through each form in an Incident Report. 3. Each arrow will become active if there is a section that exists in the direction to which the arrow points, otherwise, the arrow is grayed-out (inactive).

9.5.2.2.3 Incident Report Save and Submit Actions

 <p>Patient Info Info</p> <p>Last Name: First Name: Gender: <input type="radio"/> M <input type="radio"/> F Select DOB: Weight: Address 1: Address 2: City: State: Zip Code:</p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/></p>	<h4>Incident Report Save and Submit Actions</h4> <ol style="list-style-type: none">1. Each Incident Report Form allows the user to perform the following two actions, Save and Submit.2. Tapping Save will save the current status of the Incident Report and return you to the Main Menu.3. Tapping Submit will send the Incident Report to the Server, and return you to the Main Menu.
 <p>Store Password</p> <p>Enter Password: 123456789.....</p> <p><input type="button" value="OK"/></p>	<h4>Submitting an Incident Report</h4> <ol style="list-style-type: none">1. Prior to submission of each Incident Report, each EMS Technician will be required to enter their password.2. Enter your password in the text field and tap OK.

9.5.2.3 Patient Information Forms

9.5.2.3.1 Patient Information 1

	<h4 style="text-align: center;">Patient Information Form - Part 1</h4> <ol style="list-style-type: none"> 1. The Patient Information Form allows the user to enter in the Patients Name, Gender, Date of Birth, Weight, and Address 2. Tapping the ▼ in the lower-left-hand corner will take you to the next part of the Patient Information Form.
	<h4 style="text-align: center;">Setting the Gender</h4> <ol style="list-style-type: none"> 1. To set the Gender, simply tap on the Gender type. “M” for male, and “F” for female. 2. You can only select a Gender by using the stylus.
	<h4 style="text-align: center;">Setting the Date of Birth</h4> <ol style="list-style-type: none"> 1. To set the Date of Birth, tap on the Select DOB option on the form. 2. A Set Date Form will appear after tapping on Select DOB.
	<h4 style="text-align: center;">Set Date Of Birth Box Menu</h4> <ol style="list-style-type: none"> 1. Tap the ◀ and ▶ arrows to select the appropriate year. 2. Tap the appropriate month and day and you will be returned to the Patient Information form in which you were currently work.

9.5.2.3.2 Patient Information 2

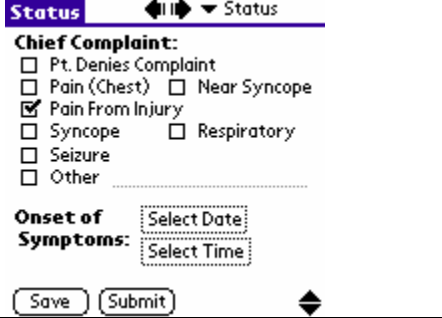
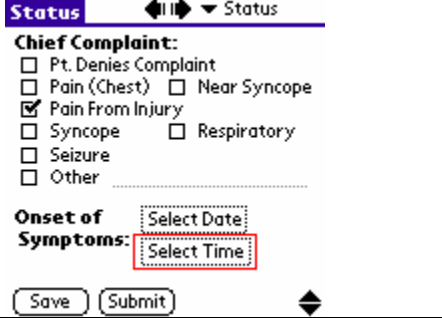
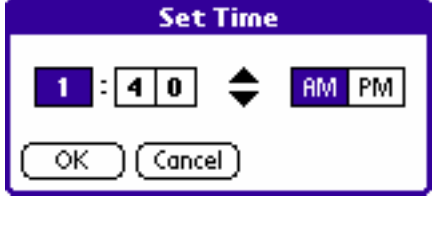
<p>Patient Info ◀▶▼ Info</p> <p><input type="checkbox"/> WPI Student</p> <p>WPI ID#: 123456789</p> <p>WPI Box#: 1234</p> <p>Patients Physician:</p> <p>First Name: Christopher</p> <p>Last Name: Elliot</p> <p>Hospital:</p> <p>Save Submit</p>	<p>Patient Information Form – Part 2</p> <ol style="list-style-type: none"> 1. The Patient Information Form allows the user to enter in the Patients Name, Gender, Date of Birth, Weight, and Address. 2. Tapping the down arrow in the lower-left-hand corner will take you to the next part of the Patient Information Form.
<p>Patient Info ◀▶▼ Info</p> <p><input type="checkbox"/> WPI Student</p> <p>WPI ID#: 123456789</p> <p>WPI Box#: 1234</p> <p>Patients Physician:</p> <p>First Name: Christopher</p> <p>Last Name: Elliot</p> <p>Hospital:</p> <p>Save Submit</p>	<p>Selecting the WPI Student Check Box</p> <ol style="list-style-type: none"> 1. If the patient is a WPI student, tap the box next to WPI Student. 2. You can only select this text box by using the stylus.

9.5.2.4 Patient Status Forms


9.5.2.4.1 Patient Status 1

<p>Status ◀▶▼ Status</p> <p>Status on Arrival:</p> <p><input type="checkbox"/> Alert <input type="checkbox"/> Verbal</p> <p><input type="checkbox"/> Pain <input checked="" type="checkbox"/> Confused</p> <p><input checked="" type="checkbox"/> Combative <input type="checkbox"/> Unconscious</p> <p><input type="checkbox"/> Arrest</p> <p>How/Where Found:</p> <p>.....</p> <p>CPR by PD BLS ALS BS</p> <p>Est Anoxic Time:</p> <p>Save Submit</p>	<p>Patient Status – Part 1</p> <ol style="list-style-type: none"> 1. This Patient Status Form allows the user to enter in the Patients Status on Arrival. Where they were found, how the CPR was performed, if any, and their Anoxic Time. 2. Tapping the ▼ in the lower-left-hand corner will take you to the next part of the Patient Status Form.
--	---

9.5.2.4.2 Patient Status 2

 <p>Status ◀▶▶ Status</p> <p>Chief Complaint:</p> <p><input type="checkbox"/> Pt. Denies Complaint</p> <p><input type="checkbox"/> Pain (Chest) <input type="checkbox"/> Near Syncope</p> <p><input checked="" type="checkbox"/> Pain From Injury</p> <p><input type="checkbox"/> Syncope <input type="checkbox"/> Respiratory</p> <p><input type="checkbox"/> Seizure</p> <p><input type="checkbox"/> Other</p> <p>Onset of Symptoms: <input type="text" value="Select Date"/> <input type="text" value="Select Time"/></p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/> ▶</p>	<p align="center">Patient Status – Part 2</p> <ol style="list-style-type: none"> 1. This Patient Status Form allows the user to enter in the Patients Chief Complaint and the date and time of their onset. 2. Tap the ^ or ▼ to go to either the previous or next section of the Patient Status Form.
 <p>Status ◀▶▶ Status</p> <p>Chief Complaint:</p> <p><input type="checkbox"/> Pt. Denies Complaint</p> <p><input type="checkbox"/> Pain (Chest) <input type="checkbox"/> Near Syncope</p> <p><input checked="" type="checkbox"/> Pain From Injury</p> <p><input type="checkbox"/> Syncope <input type="checkbox"/> Respiratory</p> <p><input type="checkbox"/> Seizure</p> <p><input type="checkbox"/> Other</p> <p>Onset of Symptoms: <input type="text" value="Select Date"/> <input type="text" value="Select Time"/></p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/> ▶</p>	<p align="center">Selecting the Time</p> <ol style="list-style-type: none"> 1. Selecting Time is similar to selecting a date. 2. Tap the Select Time box to set the time.
 <p align="center">Set Time</p> <p><input type="text" value="1"/> : <input type="text" value="40"/> <input type="text" value="AM"/> <input type="text" value="PM"/></p> <p><input type="button" value="OK"/> <input type="button" value="Cancel"/></p>	<p align="center">Setting the Time</p> <ol style="list-style-type: none"> 1. To set the time, tap the box in which you wish to modify and use the ^ and ▼ arrows to change the number in the selected box. 2. To select either AM or PM, simply tap on the appropriate box.

9.5.2.4.3 Patient Status 3

 <p>Status ◀▶▶ Status</p> <p>Location: Fuller Labs</p> <p>Skin: Warm Dry Cool Hot Moist</p> <p>Color: Pink Pale Flushed Cyanotic Ashen Normal</p> <p>Pupils: <input checked="" type="checkbox"/> Left PERL <input checked="" type="checkbox"/> Right PERL</p> <p>Cap/Refill: Normal Delayed</p> <p><input type="button" value="Save"/> <input type="button" value="Submit"/> ▶</p>	<p align="center">Patient Status – Part 3</p> <ol style="list-style-type: none"> 1. This Patient Status Form allows the user to enter in the Patients Location, Skin Condition and Color, Pupil Information, and Cap/Refill Information. 2. Tap the ^ or ▼ to go to either the previous or next section of the Patient Status Form.
---	--


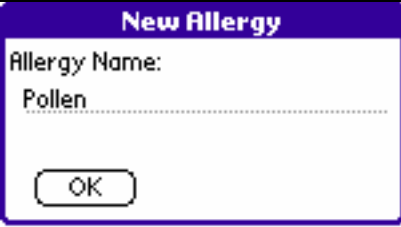

9.5.2.4.4 Patient Status 4

	<p style="text-align: center;">Patient Status – Part 4</p> <ol style="list-style-type: none"> 1. This Patient Status Form allows the user to enter in the Patients Respiratory Status. 2. Tap the ▲ to go to the previous section of the Patient Status Form.
--	---


9.5.2.5 Medications Form

	<p style="text-align: center;">Medications Form</p> <ol style="list-style-type: none"> 1. The Medication Form allows you to add New Medications through use of the New button at the bottom of the screen. Tap the New button to enter a new type of medication. 2. Tap the ▲ or ▼ to scroll through the different types of Medications.
	<p style="text-align: center;">New Medication Form</p> <ol style="list-style-type: none"> 1. Enter the name of the medication into the text field and tap the OK button.
	<p style="text-align: center;">Medications Form after Entry</p> <ol style="list-style-type: none"> 1. After medications have been entered from the New Medications Form, they are displayed in the Medication Form list.

9.5.2.6 Allergies Form

	<p style="text-align: center;">Allergies Form</p> <ol style="list-style-type: none"> 1. The Allergies Form allows you to add New Allergies through use of the New button at the bottom of the screen. Tap the New button to enter a new type of allergy. 2. Tap the ▲ or ▼ to scroll through the different types of Allergies.
	<p style="text-align: center;">New Allergy Form</p> <ol style="list-style-type: none"> 1. Enter the name of the allergy into the text field and tap the OK button.
	<p style="text-align: center;">Allergies Form after Entry</p> <ol style="list-style-type: none"> 1. After allergies have been entered from the New Allergies Form, they are displayed in the Allergies Form list.

9.5.2.7 History Form

	<p style="text-align: center;">History Form</p> <ol style="list-style-type: none"> 1. The History Form allows you to enter the patients past history.
---	--

9.5.2.8 Vitals Form

	<p style="text-align: center;">Vitals Form</p> <ol style="list-style-type: none"> 1. The Vitals Form allows you to add New Measurements through use of the New button at the bottom of the screen. Tap the New button to enter a new type of measurement. 2. Tap the ▲ or ▼ to scroll through the different Vitals.
	<p style="text-align: center;">New Measurement Form</p> <ol style="list-style-type: none"> 1. This form allows the user to enter measurements of the patient's Vital signs at the scene. Time, pulse, blood pressure, respiratory status, treatment and who treated the patient are all included in this form 2. When finished, tap the OK button.
	<p style="text-align: center;">Vitals Form after Entry</p> <ol style="list-style-type: none"> 1. After vital signs have been entered from the New Measurement Form, they are displayed in the Vitals Form list.

9.5.2.9 Call Type Form

<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: left;">Call Type:</th> </tr> </thead> <tbody> <tr> <td>Major Trauma</td> <td>Minor Trauma</td> </tr> <tr> <td>Burn</td> <td>Psych</td> </tr> <tr> <td>Major Medical</td> <td>Minor Medical</td> </tr> <tr> <td>Seizure</td> <td>OB/GYN</td> </tr> <tr> <td>Arrest</td> <td>SOB</td> </tr> <tr> <td>Chest Pain</td> <td>ETOH</td> </tr> </tbody> </table>	Call Type:		Major Trauma	Minor Trauma	Burn	Psych	Major Medical	Minor Medical	Seizure	OB/GYN	Arrest	SOB	Chest Pain	ETOH	<p style="text-align: center;">Call Type Form</p> <ol style="list-style-type: none"> 1. The Call Type Form allows you to enter the type of call that was made. 2. To select a type of call, simply tap on the appropriate type of call.
Call Type:															
Major Trauma	Minor Trauma														
Burn	Psych														
Major Medical	Minor Medical														
Seizure	OB/GYN														
Arrest	SOB														
Chest Pain	ETOH														

9.5.2.12 Responder Form

<p>Responders ◀▶ ▾ Responders</p> <p>Click to Enter Submitter ID</p> <p>Click to Enter Responder 1</p> <p>Click to Enter Responder 2</p> <p>Click to Enter Responder 3</p> <p>Click to Enter Responder 4</p> <p>Save Submit</p>	<p style="text-align: center;">Responder Form</p> <ol style="list-style-type: none"> 1. The Responder Form allows the technician to enter him/herself and up to 4 other responders that may have responded to an incident. 2. The Submitter ID is the technician sending the report, while all other responders are just those that were on scene. 3. To select a Responder, tap on the field in which you wish to add a responder.
<p style="text-align: center;">Choose a Responder</p> <p>200 L'Heureux 201 Privett 202 McHugh</p> <p style="text-align: right;">▶</p>	<p style="text-align: center;">Selecting a Responder</p> <ol style="list-style-type: none"> 1. Select from the list the Responder of your choice by tapping on that responder's name.
<p>Responders ◀▶ ▾ Responders</p> <p>202 McHugh</p> <p>201 Privett</p> <p>200 L'Heureux</p> <p>Click to Enter Responder 3</p> <p>Click to Enter Responder 4</p> <p>Save Submit</p>	<p style="text-align: center;">Responder Form after Entry</p> <ol style="list-style-type: none"> 1. Each field will list the Responder that was selected for that field.

10 Server Installation Guide

10.1 Server System Requirements

Windows NT/2000 Server or higher with access to an NT/2000 domain.

Internet Information Services

10.2 Server Application

1. Download the Java 2 Platform Standard Edition Runtime Environment from <http://java.sun.com/j2se/1.4.1/download.html>.
2. Run the downloaded installer. Agree to the license agreement and perform a “Typical” install.
3. Copy the “EMSServer” folder from the zip file to the root level of the C:\ drive.
4. Copy the “emsserverprefs” folder from the zip file to the root level of the C:\ drive.
5. Edit the following files in the “emsserverprefs” folder to update them based on your environment:

File Name	Function
dbhostname.txt	Hostname where DBMS can be found
dbusername.txt	Username with which to access DBMS
dbpassword.txt	Password with which to access DBMS
dbport.txt	MySQL port
dbname.txt	Database name within DBMS
mailfromaddress.txt	Address from which emailed reports should appear to be sent
mailtoaddresses.txt	Addresses to which to send reports
serverport.txt	Port on which to run the server application
smtphostname.txt	Hostname of SMTP server to use for sending email reports
smtpauth.txt	SMTP username and password

10.3 Database Management System

1. Download MySQL from <http://www.mysql.com>.

2. Install MySQL using a “Typical” install.
3. After MySQL installation, run the command “C:\mysql\bin\mysqld-nt.exe -install”
4. Start the MySQL Windows service in the control panel.
5. Run C:\mysql\bin\mysql.exe
6. In the MySQL window, type “. C:\emserver\setupemsdb.sql” This will set up the new “wpiems” database and configure the tables necessary for the server to run properly.
7. In the MySQL window, type “flush privileges;”
8. Note that in its default configuration, MySQL is set up to allow anyone root access to the DBMS. If you wish to change this and require a password to access the DBMS, run the command “SET PASSWORD FOR root@"%" = PASSWORD('newpassword');”
9. By default, the user created to allow the server access to the database is “emsapp” with a password of “wpi208”. To change this, run “SET PASSWORD FOR emsapp@"%" = PASSWORD('newpassword');”

10.4 Web Interface

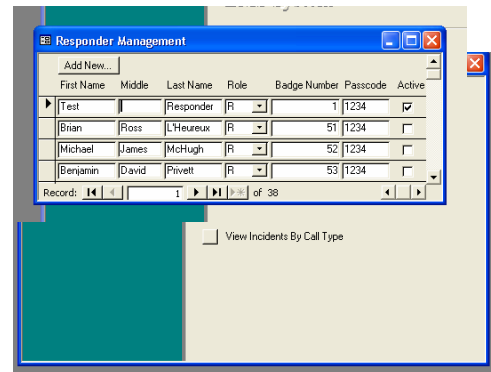
1. Download and install PHP from <http://www.php.net>. Configure it for use with Internet Information Services.
2. Set up a virtual directory to the “web-interface” directory.
3. Add two users to the NT domain: a “EMS” user and a “EMS-Admin” user. The EMS user will be able to access all generic reports without personally identifiable information. The EMS-Admin user will be the only user able to access the patient-identifiable reports.

- Remove all but the administrators and the “EMS” and “EMS-Admin” users from the permissions for the web-interface directory. Remove all but the administrators and “EMS-Admin” users from the “admin” directory in the “web-interface” directory.

10.5 Access Interface

- Install Microsoft Access 2002.
- Download the driver DLLs for MyODBC from <http://www.mysql.com/downloads/api-myodbc-2.50.html>.
- Run the installer for MyODBC, selecting the “MySQL” driver.

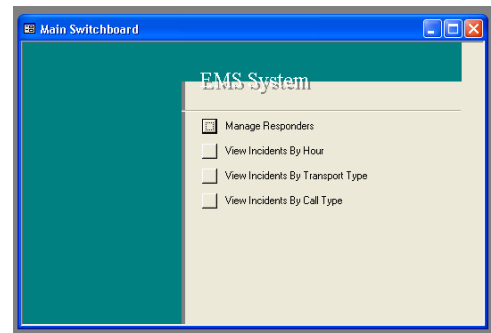
- Edit the “sample-MySQL” driver. Change the Windows DSN name to “EMS” and change the other information as appropriate to the configuration of the machine. For the username, enter “emsapp”. If Access is running on the actual server machine, type “localhost” as the hostname and leave the password blank. If the server is on a different machine, type the hostname of the machine, and set the password to the administrator password of the database, “wpiems290” by default.



11 Server Functions

11.1 Managing Responders

- The Access interface provides the ability to administer the responders that are able to submit reports via the system. To access the responder menu, open the Access database and select “Manage Responders” from the main menu.



2. The Responders window contains a list of the currently defined responders. Only responders marked “Active” are able to submit reports and be selected as a responder on the Palm application. The Passcode field is the text that the submitter must enter when submitting the report.



The screenshot shows a window titled "Responder Management" with a table of responders. The table has columns for First Name, Middle, Last Name, Role, Badge Number, Passcode, and Active. The first row is selected, showing a responder named "Test" with Role "Responder", Badge Number "1", and Passcode "1234".

First Name	Middle	Last Name	Role	Badge Number	Passcode	Active
Test		Responder	R	1	1234	<input checked="" type="checkbox"/>
Brian	Ross	L'Heureux	R	51	1234	<input type="checkbox"/>
Michael	James	McHugh	R	52	1234	<input type="checkbox"/>
Benjamin	David	Privett	R	53	1234	<input type="checkbox"/>

11.2 Viewing Reports

1. To view reports not available in the Web interface, open the Access database. Select from one of the three “View...” options, depending upon which report is desired. “View Incidents By Hour” displays the onset of symptoms time for the group of incidents in the specified date range. “View Incidents By Transport Type” displays the number of incidents of each transport type within the specified date range. “View Incidents By Call Type” displays the number of incidents of each call type within the specified date range.

11.3 Editing Data

1. The Access interface provides the ability to edit the data in all tables in the database. To access the tables directly, go to the “Window” menu and select “Unhide...”
2. Select the database, and click “OK.”
3. Select “Tables” on the left-hand shortcut bar. In the right pane, a list of the tables will appear. Most of the data that will need to be edited is in the “incident” table.